

FIGUEREDO · MALDONADO · ROSSETTO

PROGRAMAÇÃO AVANÇADA EM

MSX



**PROGRAMAÇÃO
AVANÇADA
EM
MSX**



HENRIQUE DE FIGUEREDO LUZ
MILTON MALDONADO JR.
PAULO EDUARDO GUIMARÃES ROSSETTO

PROGRAMAÇÃO
AVANÇADA
EM
MSX



© EDITORA ALEPH

EXPEDIENTE

COORDENAÇÃO EDITORIAL
COORDENAÇÃO PEDAGÓGICA
EDITORIAÇÃO
ARTE e CAPA
ILUSTRAÇÕES
PRODUÇÃO EDITORIAL

Pierluigi Piazzì
Betty Fromer Piazzì
Renato da Silva Oliveira
Ana Lúcia Antico
Odilon D. Nicoletti
Rosa Kogan Fromer

ALEPH PUBLICAÇÕES E
ASSESSORIA PEDAGÓGICA LTDA.
Caixa Postal: 20707 - Jd. Paulistano
01498 - São Paulo - SP
TEL: (011) 813-2033



Dados de Catalogação na Publicação (CIP) Internacional
(Câmara Brasileira do Livro, SP, Brasil)

L994p Luz, Henrique de Figueredo, 1967-
Programação avançada em MSX / Henrique de Figue-
redo Luz, Milton Maldonado Jr., Paulo Eduardo G. Ros-
setto. -- São Paulo : Aleph, 1986.
(Coleção MSX)

1. MSX (Computadores) Programação I. Maldonado
Júnior, Milton, 1966- II. Rossetto, Paulo Eduardo G.,
1964- III. Título. IV. Série.

86-2087

CDD-001.642

Índices para catálogo sistemático:

1. MSX : Computadores : Programação : Processamento de
dados 001.642

SUMÁRIO



	NOTA DO EDITOR	6
	PREFÁCIO	8
Capítulo 1 -	A ESTRUTURA	9
	DO BASIC MSX	
Capítulo 2 -	OPERANDO O VDP	43
	EM LINGUAGEM DE	
	MÁQUINA	
Capítulo 3 -	USOS DA	59
	IMPRESSORA	
Capítulo 4 -	OPERANDO O	81
	CASSETTE EM	
	ASSEMBLY	
Capítulo 5 -	O SISTEMA	95
	DE CARTUCHO	
Capítulo 6 -	CONSTRUINDO	113
	UM JOGO EM	
	ASSEMBLY	
APENDICES -	137

NOTA DO EDITOR

Durante muitos anos eminentes psicólogos polemizaram sobre um tema muito controvertido: a inteligência é fruto do ambiente (educação e meio familiar) ou depende de fatores genéticos (hereditariedade)?

Aparentemente os defensores da hereditariedade levaram a melhor: a inteligência parecia ser determinada basicamente pela herança genética do indivíduo. Isto gerou uma perspectiva perigosamente racista nos educadores: os indivíduos nasceriam já predestinados a se tornarem gênios ou burros em função de seu "pedigree"!

Uma crítica mais séria e científica das pesquisas que geraram esta conclusão, porém, mostrou que elas se fundamentavam em resultados duvidosos ou até em fraudes!

Modernamente determinou-se que existe sim uma influência genética mas que ela não é, de forma alguma, preponderante. Pelo contrário, a influência da educação é fundamental no desenvolvimento da inteligência individual.

O conceito racista, entretanto, está tão arraigado na cabeça dos educadores (e da população em geral) que o sistema escolar (principalmente no Brasil) não faz o menor esforço para desenvolver a capacidade intelectual dos jovens, limitando-se a abarrotá-los de informações que são posteriormente cobradas da maneira a mais imbecil possível.

Inteligência, portanto, é algo que pode (e deve!) ser ensinado na escola.

Mas como?

Não podemos simplesmente introduzir no currículo escolar algo tão vago como "aulas de Q.I."!

Uma primeira medida seria melhorar a remuneração do magistério de maneira a torná-lo uma carreira tão atraente quanto a de um médico, por exemplo.

Para se ensinar inteligência há necessidade de pessoas inteligentes e não podemos exigir que profissionais intelectualmente diferenciados se submetam à atual humilhação salarial que impera no magistério público e privado.

Uma segunda medida é a de se introduzir no currículo matérias que promovam este desenvolvimento. Antigamente este papel era feito pelo Latim, pela Matemática e pelo Xadrez. O Latim, ferramenta importantíssima no desenvolvimento da inteligência verbal, foi sumariamente extinto por imbecis que nunca entenderam sua real importância. Diga-se de passagem que para is-

so muito contribuiu o baixo nível de quem o ensinava.

A Matemática foi inteiramente deturpada pelo advento desta barbaridade chamada Matemática Moderna, mas ainda tem chance de voltar a exercer seu papel primordial, se seu ensino for completamente reestruturado. Obviamente esta mudança deverá ser feita por pessoas inteligentes e não por burocratas obtusos.

O Xadrez, que tantos benefícios trouxe ao desenvolvimento intelectual das crianças russas, por exemplo, deveria ser introduzido no currículo escolar (há propostas neste sentido no Brasil) se já não tivesse sido superado por um instrumento que o substitui com vantagens arrasadoras: o microcomputador.

Enganam-se os que pensam que o microcomputador é um instrumento educacional poderoso para se ensinar computação! Esta finalidade é um subproduto, e certamente não o mais importante, do seu uso por parte de crianças e jovens. Só quem interagiu com um microcomputador, desvendando seus mistérios, criando programas, usando "software" inteligente, sabe o quanto este instrumento é poderoso para o desenvolvimento do raciocínio e da inteligência.

Se bem empregado e implantado, o microcomputador na escola brasileira pode se tornar o instrumento básico para a emancipação intelectual da atual geração de jovens e crianças.

É claro que ele poderá também se transformar num instrumento de "burrificação" e tortura se sua implantação for deixada nas mãos dos mesmos burocratas obtusos que já conseguiram imbecilizar algumas gerações de brasileiros.

Porque dois países paupérrimos em recursos naturais como o Japão e a Itália estão entre as cinco economias mais desenvolvidas do mundo capitalista?

A resposta é simples: olhem suas escolas! Esta é a melhor (e talvez a única) saída para tirar o Brasil de seu atraso econômico e tecnológico. De nada adiantará, por exemplo, a política autoritária da SEL tentando desenvolver a tecnologia nacional por decreto! A resposta está não no "software" dos computadores mas no dos cérebros dos que os usam.

Este livro foi escrito por tres jovens brasileiros que escaparam do processo de "imbecilização" e querem, de uma forma talvez tímida mas eficiente, estender a mão para milhares de outros jovens que desejam ter acesso a este mundo maravilhoso, não da informática, mas sim da inteligência!

PREFÁCIO

Normalmente o prefácio é a parte mais chata de se ler em um livro. Quase sempre escrito depois do livro pronto, apresenta a proposta do livro e uma série de outras baboseiras. Este prefácio não vai fugir à regra, mas passaremos algumas informações preliminares para que o livro não se torne mais chato que o prefácio.

Tentamos com este livro transmitir a você, uma máquina de Von Neumann (quem leu a nota do editor do Aprofundando-se deve ter entendido), uma série de macêtes nem sempre fáceis de desvendar.

Partimos do pressuposto que o leitor deste livro já conheça, pelo menos, o "feijão com arroz" da Linguagem de Máquina e razoavelmente o sistema MSX.

Alguns programas são exemplos bem didáticos e simples que não apresentam uma aplicação muito grande, pois são destinados apenas ao aprendizado: a partir deles você poderá construir programas realmente úteis e complexos. Outros programas apresentam aplicações imediatas, como, por exemplo, os programas de cópia gráfica do Capítulo 3.

Todos os programas foram escritos no Coral ASM versão 2.1. Para que você possa entendê-los melhor, adaptá-los ao seu compilador (que pode tranquilamente ser outro) ou se você não possui um compilador Assembly, dê uma olhada no Apêndice IV.

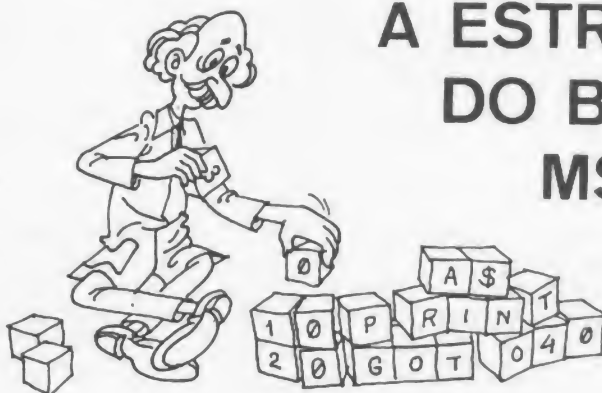
Esperamos que você goste, aprenda e que principalmente utilize as informações deste livro, obtidas, às vezes, graças a muitas madrugadas "morcegando" na frente do computador.

Finalizando, gostaríamos de agradecer a algumas pessoas que direta ou indiretamente nos deram uma força para escrever esse livro.

Pierluigi
Renato
Fernando Grossi
Fernanda e Kaduoka
Luis Alberto Castrucci

Henrique
Milton e
Paulo

A ESTRUTURA DO BASIC MSX



Vamos analisar agora de uma maneira profunda a organização de um programa BASIC na memória do micro. Normalmente a área de programa começa no endereço &H8001 e o byte &H8000 contém 00 para indicar ao Interpretador que, a partir daí, pode começar uma linha em BASIC.

Embora seja normalmente área de programa BASIC, esta configuração pode ser alterada mudando-se algumas variáveis do sistema, como veremos mais à frente.

TOKENS

Como você já deve saber, para cada palavra reservada do BASIC existe um código correspondente chamado de "token". O armazenamento de programas na forma de tokens é muito útil para a economia de memória e de tempo. Afinal, o micro só tem que reconhecer um byte ao interpretar um programa e não toda a sequência de códigos ASCII que o representaria.

Vamos imaginar o armazenamento de um comando ou de uma função pelo seu código ASCII. Por exemplo, a linha em BASIC

```
X=VARPTR(A)
```

ficaria armazenada conforme mostra a figura 1.1 .

FIGURA 1.1 - Exemplo de uma linha BASIC no formato ASCII

```
58 X
3D =
56 V
41 A
52 R
50 P
54 T
52 R
28 (
41 A
29 )
```

Sem contar o número de linha, um byte para dizer que a linha terminou e outras coisas mais, a linha exemplificada na figura 1.1 ocuparia 11 bytes. Com o sistema de tokens, porém, ela estaria armazenada como mostra a figura 1.2.

FIGURA 1.2 - Linha hipotética tokenizada

```
58 X
EF token do =
E7 token do VARPTR
28 (
41 A
29 )
```

Como você pode observar, a economia de memória é notável em relação a uma linha armazenada no formato ASCII. Quanto à economia de tempo, imagine o trabalho realizado pelo Interpretador ao executar um programa: ter que comparar letra por letra da função VARPTR (que é uma das funções do BASIC de nome mais comprido) para depois descobrir que deve saltar para a rotina que executa o VARPTR propriamente dito.

Com o sistema de tokens, o Interpretador já reconhece o &HE7 como a função VARPTR e salta para a rotina correspondente.

A tabela da ROM que indica as palavras que pertencem ao BASIC MSX começa no endereço &H3A72 e vai até o endereço &H3D22. As palavras que começam pela mesma letra estão dispostas em grupos colocados em ordem alfabética. Para demarcar uma separação entre os grupos existe um byte zerado.

Há uma tabela de 26 elementos, que se inicia a partir do endereço &H3A3E que contém, na forma LSB e MSB, o endereço no qual se inicia cada um dos grupos de tokens que começam com uma determinada letra. Por

exemplo, o primeiro elemento da tabela (bytes &H3A3E e &H3A3F) aponta para o endereço &H3A72, que é onde se inicia a tabela das palavras reservadas iniciadas com A.

Já que todas as palavras de um mesmo grupo começam com a mesma letra, torna-se desnecessário armazenar a primeira letra de todas elas. Para indicar o fim dos códigos ASCII de uma determinada palavra, sua última letra estará sempre com o bit 7 setado, sendo que o byte seguinte indicará o código da token. A figura 1.3 mostra o armazenamento da função VARPTR na ROM do micro.

FIGURA 1.3 - Armazenamento da função VARPTR

```

XX
XX
41 A
52 R
50 P
54 T
D2 R (com bit 7 setado)
E7 código da TOKEN
XX
XX

```

Vamos apresentar, na figura 1.4 um programa Assembly que, a partir dos códigos ASCII de uma palavra reservada do BASIC, obtém, a token correspondente, através de uma varredura da ROM.

FIGURA 1.4 - Programa TOKEN

	ORG	0C000H	
00B4	GINLIN=	00B4H	
00A2	CHPUT=	00A2H	
C000 CDB400	CALL	GINLIN	RECEBE PALAVRA
C003 23	INC	HL	APONTA P/ 1ª LETRA
C004 7E	LD	A,(HL)	A CONTEM 1ª LETRA
C005 23	INC	HL	APONTA P/ 2ª LETRA
C006 22B6C0	LD	(PALAVR),HL	GUARDA END QUE APONTA PARA 2ª LETRA
C009 0641	SUB	'A'	LOCALIZA ENDEREÇO
C00B 87	ADD	A,A	QUE APONTA PARA
C00C 0600	LD	B,0	1º BLOCO DE PALAVRAS
C00E 4F	LD	C,A	RESERVADAS INICIADAS
C00F 113E3A	LD	DE,3A3EH	COM A MESMA LETRA.
C012 EB	EX	DE,HL	
C013 09	ADD	HL,BC	
C014 EB	EX	DE,HL	
C015 1A	LD	A,(DE)	FAZ DE APONTAR P/
C016 4F	LD	C,A	BLOCO DE PALAVRAS
C017 13	INC	DE	RESERVADAS INICI
C018 1A	LD	A,(DE)	ADAS COM A MESMA
C019 57	LD	D,A	LETRA.
C01A 59	LD	E,C	
C01B 1A	LD	A,(DE)	PEGA LETRA DA TABELA
C01C FE00	CP	0	NA ROM
			VERIFICA SE E FIM

C01E 2843	JR	Z,ERRO	7DE BLOCO
C020 C8BF	RES	7,A	7SE AFIRM. SALTA
C022 8E	CP	(HL)	7PARA ERRO
			7RESSETA 0 BIT 7.
			7COMPARA COM LETRA DA
			7PALAVRA A SER
			7PESQUISADA.
C023 2009	JR	NZ,PROXPA	7SE () SALTA P/ PROXPA.
C025 1A	LD	A,(DE)	7VERIFICA SE E
C026 C87F	BIT	7,A	7FINAL DE PALAVRA
C028 2013	JR	NZ,IMPTKN	7SE FOR SALTA P/ IMPTKN.
C02A 23	INC	HL	7INCREMENTA HL E DE.
C02B 13	INC	DE	7P/ COMPARAR PROX LETRA
C02C 18ED	JR	LOOP	7EFETUA LOOP.
C02E			
C02E 2A8C0	PROXPA:	LD HL,(PALAVR)	7HL APONTA P/ 23
C031 1A		LD A,(DE)	7LETRA E DE
C032 C87F		BIT 7,A	7APONTA PARA
C034 2804		JR Z, PROC	7PROXIMA PALAVRA
C036 13		INC DE	7RESERVADA.
C037 13		INC DE	
C038 18E1		JR LOOP	
C03A	PROC:		
C03A 13		INC DE	
C03B 18F1		JR PROXPA	
C03D	IMPTKN:		
C03D 13		INC DE	7APONTA DE P/ TOKEN.
C03E 1A		LD A,(DE)	7CARREGA A COM TOKEN.
C03F E6F0		AND 11110000B	7PEGA 19 NIBBLE.
C041 1F		RRA	7ROTACIONA
C042 1F		RRA	7BITS.
C043 1F		RRA	
C044 1F		RRA	
C045 C630		ADD A,30H	7SOMA 30H PARA A CONTER
			7O CODIGO DO N2
C047 FE3A	CP	3AH	7VERIFICA SE E LETRA.
C049 3802	JR	C,IMPNI8	7SE NAO,IMPRIME.
C04B C607	ADD	A,7	7SOMA 7 P/ A CONTER
			7CODIGO DE LETRA
C04D	IMPNI8:		
C04D CDA200		CALL CHPUT	7IMPRIME DIGITO.
C050 1A		LD A,(DE)	7PEGA COD TOKEN.
C051 E60F		AND 00001111B	7PEGA 29 NIBBLE.
C053 C630		ADD A,30H	7SOMA 30H PARA A CONTER
			7O CODIGO DO N2
C055 FE3A	CP	3AH	7VERIFICA E LETRA.
C057 3802	JR	C,IMP2NI	7SE NAO IMPRIME
C059 C607	ADD	A,7	7SOMA 7 P/ A CONTER
			7O CODIGO DO N2
C05B	IMP2NI:		
C05B CDA200		CALL CHPUT	7IMPRIME DIGITO.
C05E 00		NOP	7FUTURA EXPANSAO.
C05F 00		NOP	7FUTURA EXPANSAO.
C060 00		NOP	7FUTURA EXPANSAO.
C061 00		NOP	7FUTURA EXPANSAO.
C062 C9		RET	7RETORNA AO BASIC.
C063	ERRO:		
C063 0615		LD B,21	7CASO TOKEN NAO
C065 2170C0		LD HL,MENS	7ENCONTRADA,
C068	LOOP2:		7IMPRIME MENSAGEM
C068 7E		LD A,(HL)	
C069 CDA200		CALL CHPUT	7DE ERRO.
C06C 23		INC HL	
C06D 10F9		DJNZ LOOP2	
C06F C9		RET	
C070 50414C41	MENS:	DEFB 'PALAVRA NAO RESERVADA'	
C074 56524120			
C078 4E414F20			
C07C 52455345			
C080 52564144			
C084 41			
C085 00	TOKEN:	DEFB 0	
C086 0000	PALAVR:	DEFW 0	
C088		END	

TOKEN DE FUNÇÃO

Quando a palavra reservada não corresponde a um comando, mas sim a uma função, ela recebe um tratamento especial:

Se você notar bem, a maioria das tokens de função apresentam o seu código entre 1 e 48. Em uma linha de programa, essas funções não estão armazenadas apenas com a token da tabela de palavras reservadas; elas são precedidas de um byte contendo &HFF, sendo que também o bit 7 do byte da token fica setado.

Ou seja, o código que você obtém na tabela não é imediatamente o código que está armazenado num programa. Para obtê-lo você terá que setar o bit 7 (ou somar &H80). Vejamos um exemplo na figura 1.5.

FIGURA 1.5 - Linha de programa com token de função

```
10 PRINT SIN(A)

xx]epl (endereço da próxima linha)
xx
00 n1 (número da linha)
0A]
91 PRINT
FF identificador de função
89 token do SIN
28 (
41 A
29 )
00 fl (fim de linha)
```

Ao consultarmos a tabela da ROM, constatamos que o código da token da função SIN é 09, e subtraindo &H80 do &H89 (ou resetando o bit 7 desse byte) obtemos o mesmo código.

ESTRUTURA DE UMA LINHA DE PROGRAMA

A maneira pela qual o micro armazena uma linha de programa é bastante simples:

-Os dois primeiros bytes contêm o endereço da próxima linha.

-Os dois bytes seguintes representam na forma LSB e MSB o número da linha. É por isso que uma linha só pode ter um número entre 0 e 65529.

-A partir desse byte podem existir mais 254 bytes para os comandos do BASIC, e para indicar o final da linha existe um byte zerado. A figura 1.6 mostra a estrutura de uma linha "simples" de BASIC.

FIGURA 1.6 - Exemplo de uma linha BASIC

```
10 PRINT A
09 ] epl
80 ]
0A ] nl
00 ]
91 PRINT
41 A
00 fl
00 ] fim de programa
00
```

É importante ressaltar que ao final de um programa BASIC existem dois bytes zerados, indicando para o interpretador que o programa termina nesse ponto.

ESCONDENDO LINHAS

Você pode esconder uma linha de programa do LIST, bastando para isso alterar seu epl (endereço da próxima linha), fazendo-o apontar para o início (epl) da próxima linha que você quer que apareça.

Vejamos o programa exemplo da figura 1.7 .

FIGURA 1.7 - Exemplo de programa BASIC

```
10 A=0
20 PRINT"EU ESTOU AQUI!!!"
30 PRINT"MAS NÃO APAREÇO NA LISTAGEM"
40 LIST
```

Com um dump da área de programa, você obteria os dados da figura 1.8 .

Suponhamos que você queira esconder as linhas 20 e 30 do programa.

-Para isso, vamos alterar o epl (endereço da próxima linha) da linha 10, fazendo-o apontar para o início da linha 40 que é a primeira linha que aparecerá depois daquelas escondidas.

-Usando um monitor, vamos alterar o byte &H8001 de &H0A para &H45. Agora o epl da linha 10 aponta para a linha 40.

-Execute o programa com um RUN ou um GOTO 10.

FIGURA 1.8 - Dump de memória

8000	00	0A	80	0A	epl da linha 10
8004	00	41	EF	0F	..A..	número da linha 10
8008	0A	00	22	80	.."	epl da linha 20
800C	14	00	91	22"	número da linha 20
8010	45	55	20	45	EU E	
8014	53	54	4F	55	STOU	
8018	20	41	51	55	AQU	
801C	49	21	21	21	I!!!	
8020	22	00	45	80	..E..	epl da linha 30
8024	1E	00	91	22"	número da linha 30
8028	4D	41	53	20	MAS	
802C	4E	B0	4F	20	N00	
8030	41	50	41	52	APAR	
8034	45	80	4F	20	E..0	
8038	4E	41	20	4C	NA L	
803C	49	53	54	41	ISTA	
8040	47	45	4D	22	GEM"	
8044	00	4B	80	28	..K..(epl da linha 40
8048	00	93	00	00	número da linha 40
804C	00					Fim de Programa

Você verá que as linhas 20 e 30 são executadas mas não aparecem na listagem.

Quando o programa é executado, as linhas 20 e 30 são executadas normalmente, pois durante a execução o Interpretador não consulta o epl das linhas do programa. Mas quando certos comandos como o LIST, LLIST ou o SAVE (para programas em ASCII) por exemplo são executados, o epl das linhas é consultado e elas não são notadas pelo Interpretador.

As linhas 20 e 30 podem, facilmente voltar a aparecer. Basta inserir uma linha com numeração inferior às que foram escondidas, pois ao fazer isso o sistema operacional reajusta todos os epl do programa.

TRATAMENTO DE NÚMEROS

Assim como o BASIC MSX guarda as palavras reservadas na forma de tokens (para ocupar menos espaço e economizar tempo de processamento) também guarda os números de uma maneira toda especial.

O BASIC permite 3 tipos de variáveis numéricas:

1) As inteiras representadas pelo símbolo "%" (por cento) após o nome da variável ou pela função DEFINT, às quais podemos atribuir valores inteiros na faixa de -32768 a 32767, e que ocupam apenas dois bytes de memória.

2) As de simples precisão, identificadas pelo ponto de exclamação (!) ou pelo comando DEFSNG. Elas podem assumir valores de até seis dígitos, com um expoente que vai de -64 a +62, ocupando 4 bytes de memória.

3) As de dupla precisão, representadas pelo "number" (#) ou pelo comando DEFDBL. Elas podem representar números de até 14 dígitos com um expoente de -64 a +62, consumindo oito bytes de memória (a menos que você esteja realizando cálculos altamente precisos isto pode representar um desperdício!).

Se você não definir no seu programa se as variáveis serão inteiras, simples ou duplas, o Interpretador assumirá todas como sendo de dupla precisão.

NÚMEROS EM LINHAS DE PROGRAMAS

Quando uma linha é digitada, o BASIC não se preocupa em saber se a variável é inteira, simples ou dupla. O armazenamento do número na linha é feito de forma a economizar o máximo possível de memória, numa espécie de "tokenização" dos números na memória; ou seja, tal qual as palavras reservadas do BASIC, nem sempre o que você vê na tela do micro com um LIST é realmente, o que está na memória.

REFERÊNCIA A LINHAS

Durante a "tokenização" de uma linha, todos os números que se referem à linhas de programa são armazenados de duas maneiras. Por exemplo, a linha

```
10 GOTO 20000
```

Quando digitada será armazenada como mostra a figura 1.9.

FIGURA 1.9 - Tokenização de um número

```

xx] epl (endereço da próxima linha)
xx]
00] n1 (número da linha)
0A]
09 GOTO
20 "espaço"
0E id
20] número (LSB e MSB)
4E]
00 fl

```

O byte contendo um &H0E (id) serve para indicar que os dois bytes a seguir se referem a uma linha de programa e estão armazenados na forma LSB e MSB.

Quando a linha for executada pela primeira vez, o Interpretador alterará o byte de identificação para &H0D e os dois bytes seguintes conterão o endereço de início da linha em questão. Isto é feito para agilizar a execução nas próximas vezes. Essa técnica de atualização e otimização dos números referentes a linhas de programas é conhecida como Apontadores Progressivos.

TRATAMENTO DE NÚMEROS INTEIROS

Os números inteiros possuem um armazenamento bastante peculiar, que é dividido em três estágios distintos:

- * Números inteiros entre 0 e 9
- * Números inteiros entre 10 e 255
- * Números inteiros entre 256 e 32767

Vamos analisar a seguinte linha de programa.

```
10 A=5
```

Examinando os bytes na memória do micro teremos os dados da figura 1.10

O Interpretador reconhece o byte com &H16 como sendo o "5". Esse byte é uma espécie de token para armazenar um número de um dígito. Os números de um dígito e suas respectivas tokens estão listados na figura 1.11.

Os números negativos são precedidos pela token do "-", que é o &HF2. Vejamos, na figura 1.12, como ficaria a mesma linha se quiséssemos armazenar número -5.

FIGURA 1.10 - Dump de uma linha de programa

```

xx]ep1
xx]
00]n1
0A]
41  A
EF  =
16  token do 5
00  f1

```

FIGURA 1.11 - Tokens para números de um dígito

TOKEN	DÍGITO	TOKEN	DÍGITO
&H11	0	&H16	5
&H12	1	&H17	6
&H13	2	&H18	7
&H14	3	&H19	8
&H15	4	&H1A	9

FIGURA 1.12 - Dump de uma linha de programa

```

xx]ep1
xx]
00]n1
0A]
41  A
EF  =
F2  token do -
16  token do 5
00  f1

```

Vejam os agora como o micro armazena um número entre 10 e 255.

Para reconhecer que determinado número está nessa faixa, o micro coloca um byte de identificação, que no caso é o &H0F. Vamos analisar uma linha que apresente tal armazenamento:

```
10 S=30
```

A figura 1.13 mostra mais detalhadamente, a forma de armazenamento da linha:

FIGURA 1.13 - Dump de uma linha de programa

```

xx]epl
xx]
00]n1
0A]
53 S
EF =
0F id
2E 30
00 fl

```

Como você pode notar, o &H2E é 30 em decimal, e isso se repete para os demais bytes até 255.

Os números inteiros restantes, que vão de 256 a 32767 possuem outra forma de armazenamento. Existe também um byte de identificação (&H1C) precedendo os bytes que compõem o número. O número em si é armazenado em dois bytes, na forma LSB e MSB (ou parte baixa e parte alta) Por exemplo, a linha de programa:

```
10 X=12345
```

é armazenada como mostra a figura 1.14 :

FIGURA 1.14 - Armazenamento de um número na forma inteira

```

xx]epl
xx]
00]n1
0A]
58 X
EF =
1C id
30 48*256=12288
39 57*1 = 57      12288+57=12345
00 fl

```

NÚMEROS EM SIMPLES PRECISÃO

Em uma linha de programa, os números que possuem de uma a seis casas decimais, ou forem maiores que 32767 serão armazenados no formato de simples precisão, o qual ocupa cinco bytes. O primeiro é um byte de identificação da forma de armazenamento do número, que no caso conterà &H1D. O segundo byte indica o expoente, e os três bytes seguintes armazenam os dígitos do número.

Por exemplo, a linha

10 MA=1.23456

estará na memória como mostra a figura 1.15 :

FIGURA 1.15 - Armazenamento em simples precisão

```
xx] ep1
xx]
00] n1
0A]
4D M
41 A
EF =
1D id
41 byte de expoente
12] dígitos
34]
56]
00 fl
```

Repare o primeiro byte do dígito. O valor dele é &H12 e os dois primeiros dígitos do número são 1 e 2 !!!

Na hora de armazenar a linha, o Interpretador transforma o código ASCII do primeiro dígito no próprio número e, como estamos lidando com números decimais, um dígito só vai de zero a nove (que em binário vai do &B0000 a &B1001).

Note que o maior dígito que temos é o nove (&H1001), que só ocupa quatro bits. Então, por que não colocar dois dígitos (cada um ocupando quatro bits) em apenas um byte.

Assim o primeiro "nibble" (conjunto de quatro bits) representa o primeiro dígito, o segundo "nibble" o segundo dígito, e assim sucessivamente até o término do número (e dos três bytes).

NÚMEROS EM DUPLA PRECISÃO

O armazenamento de um número que possua mais de 6 dígitos é feito em dupla precisão. O sistema de armazenamento na linha de programa é igual ao de simples precisão. As únicas diferenças são o byte de identificação (que no caso é o &H1F) e o número de bytes para armazenar dois dígitos, que agora são sete.

Vejamos, na figura 1.16 , como é armazenado o número:

10 PI=3.141592657

FIGURA 1.16 - Armazenamento em dupla precisão

```
xx] epl
xx]
00] nl
0A]
50 P
49 I
EF =
1F id
41 byte de expoente
31 dígitos
41
59
26
57
00
00 fl
```

BYTE DE EXPOENTE

Quando foram criados os MSX, resolveu-se que apenas um byte seria dedicado ao armazenamento do expoente. Esse byte teria que dizer também se o número é positivo ou negativo, por isso seu bit sete foi reservado para dizer o sinal do número.

Com isso restam sete bits (0-6), o que representa 128 posições (0-127) para indicar um expoente que pode ser positivo ou negativo.

Vamos dividir por dois essas 128 posições. (metade para os expoentes negativos e metade para os expoentes positivos).

Levando em consideração só os sete primeiros bits do byte de expoente, a organização ficaria da seguinte maneira:

&H01-&H3F - Expoente negativo (-63 a -1)

&H40-&H7F - Expoente positivo (0 a 63)

Depois de ler tudo isso, você pensa:

Se o byte de expoente pode representar 64 expoentes diferentes, como é que o expoente máximo que eu consigo representar no meu micro é 62 ?

O que acontece é que, quando o micro imprime um número em simples ou dupla precisão, se ele ocupar mais de 13 dígitos, ele é impresso em notação científica e neste caso aparece o expoente na impressão. Um número em notação científica apresenta sempre o primeiro dígito como um inteiro, e os dígitos restantes são colocados depois do ponto.

O número que você retira do byte de expoente não é igual ao expoente de um número em notação científica, embora o micro assim os imprima. O Interpretador considera como se todos os dígitos estivessem à direita do ponto. Por exemplo, o número

.05

é representado em notação científica como

5E-2

mas, na notação interna do micro, o mesmo número estaria da seguinte forma:

.5E-1

Então, quando o micro imprime um expoente 62, na sua notação interna o expoente é 63.

A mesma coisa acontece com os números de expoente negativo. Enquanto você vê no vídeo um número do tipo 3.14984E-64, o expoente está armazenado como se ele estivesse na forma .314984-63.

Note que, com esse tipo de notação o número .5 possui o expoente zero que está no grupo dos expoentes positivos, enquanto que em notação científica ele possuiria um expoente negativo (-1).

Embora isso só aconteça na área de variáveis, quando o byte de expoente está zerado o Interpretador assume o valor da variável como sendo zero, mas os bytes que indicam os algarismos não são destruídos.

NÚMEROS EM OUTRAS BASES

Em uma linha de programa um número também pode estar nas bases hexadecimal, binária ou octal. O Interpretador também possui uma maneira especial de armazenar esses números.

Vejamos seguinte linha de programa:

```
10 A=&HA023+&B1010+&H37
```

Seu armazenamento "tokenizado" na memória está mostrado pela figura 1.17

FIGURA 1.17 - Linha de programa com número em outras bases

```

xx] epl
xx]
00] nl
0A]
41] A
EF] token do =
0C] id hexadecimal
23] número (LSB e MSB)
A0]
F1] token do +
26] &
42] B
31] i
30] 0
31] i
30] 0
F1] +
0B] id octal
1F] número (LSB e MSB)
00]
00] fl

```

Note que existe um byte de identificação para os números em hexadecimal (&H0C) e outro para os números octais (&H0B). O valor dos números em si é armazenado na forma LSB e MSB.

Já os números binários são armazenados no formato ASCII, e os caracteres "&" e "B" já servem como bytes de identificação dessa forma de armazenamento.

ÁREA DE VARIÁVEIS

Há uma área na memória do micro que é reservada para o armazenamento das variáveis de um programa BASIC. O início dessa área é dado pela variável do sistema VARTAB (&HF6C2), e normalmente aponta para o final do seu programa BASIC. Uma outra variável de sistema, a STREND (&HF6C6), indica o final dessa área. Sempre que uma variável do BASIC é consultada, o Interpretador vasculha a área de memória delimitada por VARTAB e STREND e, se a variável não estiver definida naquela área, o Interpretador assume o seu valor como sendo zero (variáveis numéricas) ou uma string vazia.

O valor da STREND se iguala ao da VARTAB sempre que uma nova linha BASIC é introduzida no seu programa. Devido a esse fato, quando uma linha de programa é introduzida, todos os valores de variáveis que você tenha armazenado são perdidos.

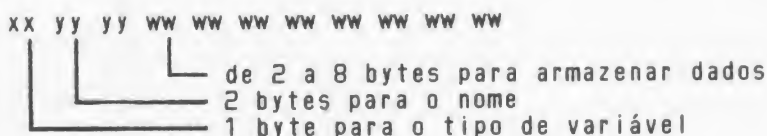
Existem quatro tipos de variáveis que são armazenadas nessa área.

São elas:

Inteiras	ocupam 2 bytes
strings	ocupam 3 bytes
precisão simples	ocupam 4 bytes
precisão dupla	ocupam 8 bytes

Uma variável possui a sintaxe de armazenamento apresentada na figura 1.18 .

FIGURA 1.18 - Forma de armazenamento de variáveis



O primeiro byte indica o tipo de variável que está armazenada, e o seu código também já indica em quantos bytes estão armazenados os dígitos.

Por exemplo, se o primeiro byte contiver o valor 2, saberemos que a variável é inteira e que o seu valor está armazenado em dois bytes.

As variáveis vão sendo armazenadas à medida em que forem sendo definidas pelo programa.

O micro assume como default, para as variáveis sem os indicadores de tipo (% , # , !), o armazenamento em dupla precisão. Isso sem se importar com o valor da variável em si. Se o seu programa não possuir cálculos altamente precisos (como um programa de astronomia por exemplo) é melhor utilizar variáveis inteiras, que são processadas mais rapidamente e ocupam menos memória. Você pode selecionar através das funções DEFINT, DEFSNG e DEFDBL, o tipo da variável que começa com uma determinada letra. Assim não haverá necessidade de colocar, após cada variável, o indicador de tipo.

Os comandos DEFINT, DEFSTR, DEFSNG e DEFDBL alteram uma tabela que tem início no endereço &HF6CA (DEFTBL), a qual possui 26 elementos, um para cada letra do alfabeto.

Cada elemento indica o tipo de uma variável sem indicador de tipo que foi encontrada no programa durante a execução.

Para cada tipo de variável, o elemento pode assumir os seguintes valores:

02	inteira
03	string
04	simples
08	dupla

Quando é consultado ou alterado o valor de uma variável que não apresenta indicador de tipo, o Interpretador consulta a tabela para saber qual o tipo que deve assumir para aquela variável.

Por exemplo, você pode ter três variáveis chamadas AB, uma de cada tipo.

Se o primeiro elemento da tabela DEFTBL for 8 (que é default) você só poderá acessar os valores das variáveis AB inteira, de precisão simples e strings se forem usados os indicadores de tipo logo após o nome da variável.

Digite os comandos:

```
AB% = 70
AB! = 53.56
AB# = 30.567765291456
AB$ = "AB"
```

O valor do primeiro elemento da DEFTBL é 8, portanto se você comandar

```
PRINT AB
```

como resultado você obterá:

```
30.567765291456
```

Agora comande:

```
POKE &HF6CA,2
```

e imprima o valor da variável AB com o comando

```
PRINT AB
```

Você obteve como resultado:

```
70
```

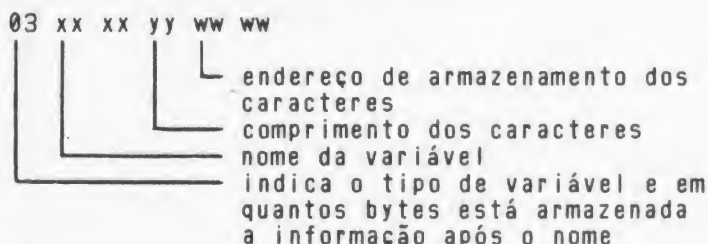
Quando o POKE foi executado, houve alteração no valor do primeiro elemento da tabela DEFTBL, que indica o tipo de todas as variáveis que começam com a letra A e foram usadas sem o respectivo indicador de tipo. Como o valor "pokeado" foi 2, foi impresso o valor da variável inteira.

Se alterássemos o valor para 4 ou 3, seria impresso o valor da variável de simples precisão ou da variável string, respectivamente.

VARIÁVEIS TIPO STRING

As variáveis do tipo string não armazenam na área de variáveis a sequência dos caracteres. Por isso elas possuem a sintaxe de armazenamento apresentada na figura 1.19.

FIGURA 1.19 - Armazenamento de variáveis tipo string



A Variável de Sistema FRETOP (contém &HF168 como default) armazena o endereço que vai receber o último caráter da primeira string definida, se durante a execução for feita alguma operação envolvendo as strings. Portanto, se a string possuir 10 caracteres, o endereço &HF15F conterá o primeiro caráter da string. A segunda string definida terá seu último caráter ocupando o endereço &HF15E e assim sucessivamente.

Se num programa houver uma linha com a seguinte sintaxe

```
130 A$="IUAHCFEFAKOUAK"
```

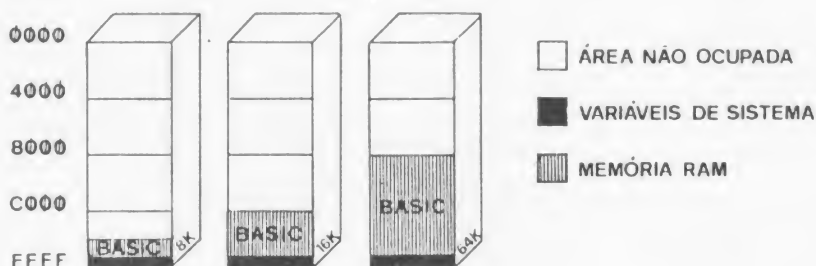
ao executá-la, o Interpretador criará a variável com o devido nome na área de variáveis, mas o endereço para o qual ela vai apontar, estará na área de programa, justamente na linha BASIC, indicando o primeiro caráter da string.

Mas se for feita alguma operação com essa string, como por exemplo, somá-la com outra string, os textos das duas strings serão transferidos para a área de strings e a variável apontará para esse endereço.

RELOCANDO PROGRAMAS EM BASIC

O Sistema MSX exige uma RAM mínima de 8 Kbytes e o topo da memória sempre deve conter RAM, pois ali estão as variáveis de sistema. Se o micro dispuser de mais memória, como é o caso dos MSX brasileiros, esta é alocada até que se chegue ao endereço zero. A figura 1.20 apresenta esquematicamente várias estruturas possíveis de RAM.

FIGURA 1.20 - Possíveis configurações de memória



Em um MSX com 16 Kbytes de RAM, a área reservada para o programa BASIC começaria no endereço &HC000.

Para indicar o endereço onde se inicia a RAM disponível existe a variável **BOTTOM** (&HFC48) e para indicar o início do texto BASIC existe a variável **TXTTAB**(&HF676).

Existem mais 3 variáveis de sistema que obrigatoriamente também precisam ser alteradas para que se consiga relocar o BASIC. Essas variáveis controlam a área de variáveis do BASIC. São elas:

```
VARTAB  &HF6C2
ARYTAB  &HF6C4
STREND  &HF6C6
```

Quando um programa está funcionando em outra área inicial de memória que não o &HB000, todas as funções, como o carregamento e a gravação de um programa, não são alteradas.

O Programa da figura 1.21 divide a RAM em duas partes distintas (&HB000-&HBFFF e &HC000-valor do HIMEM) e colocaremos em cada uma delas um programa BASIC.

FIGURA 1.21 - Programa RELOC

```

                                ORG 0C000H
F676      TXTTAB:EQU 0F676H
F6C2      VARTAB:EQU 0F6C2H
F6C4      ARYTAB:EQU 0F6C4H
F6C6      STREND:EQU 0F6C6H
FC48      BOTTOM:EQU 0FC48H
C000 2100B0      LD HL,0B000H      ;INICIALIZA A NOVA
C003 0603      LD B,3              ;AREA PARA O BASIC
C005      LOOP:                                ;COM 3 ZEROS.
C005 3600      LD (HL),0
C007 23      INC HL
C008 10FB      DJNZ LOOP
C00A 2100B0      LD HL,0B000H      ;APONTA AS VARIÁVEIS
C00D 22CAFF      LD (VAR),HL      ;DE SISTEMA ALTERNA-
C010 22CCFF      LD (ARY),HL      ;TIVAS PARA A NOVA
C013 22CEFF      LD (STR),HL      ;AREA DO BASIC.
C016 22D2FF      LD (BTM),HL
C019 23      INC HL
C01A 22D0FF      LD (TXT),HL
C01D C9      RET                  ;RETORNA AO BASIC.
C01E      FLFLP:
C01E ED4BCAFF      LD BC,(VAR)      ;TROCA O VALOR DE
C022 2AC2F6      LD HL,(VARTAB)      ;VARTAB COM O
C025 ED43C2F6      LD (VARTAB),BC      ;DE VAR.
C029 22CAFF      LD (VAR),HL
C02C ED4BC0FF      LD BC,(ARY)      ;TROCA O VALOR DE
C030 2AC4F6      LD HL,(ARYTAB)      ;ARYTAB COM O
C033 ED43C4F6      LD (ARYTAB),BC      ;DE ARY.
C037 22CCFF      LD (ARY),HL
C03A ED4BC0FF      LD BC,(STR)      ;TROCA O VALOR DE
C03E 2AC6F6      LD HL,(STREND)      ;STREND COM O
C041 ED43C6F6      LD (STREND),BC      ;DE STR.
C045 22CEFF      LD (STR),HL
C048 ED4BD0FF      LD BC,(TXT)      ;TROCA O VALOR DE
C04C 2A76F6      LD HL,(TXTTAB)      ;TXTTAB COM O
C04F ED4376F6      LD (TXTTAB),BC      ;DE TXT.
C053 22D0FF      LD (TXT),HL
C056 ED4BD2FF      LD BC,(BTM)
C05A 2A49FC      LD HL,(BOTTOM)      ;TROCA O VALOR DE
C05D ED4348FC      LD (BOTTOM),BC      ;BOTTOM COM O
C061 22D2FF      LD (BTM),HL      ;DE BTM.
C064 C9      RET
FFCA      VAR: EQU 0FFCAH      ;O VALOR DAS
FFCC      ARY: EQU 0FFCCH      ;VARS. DE SISTEMA
FFCE      STR: EQU 0FFCEH      ;FICA ARMAZENADO
FFD0      TXT: EQU 0FFD0H      ;EM AREA NAO
FFD2      BTM: EQU 0FFD2H      ;USADA PELO
                                ;SISTEMA
C065      END

```

O primeiro bloco do programa inicializa a área de memória a partir do &HB000 para receber o segundo programa BASIC.

O segundo bloco, a partir do LABEL FLFLP, altera as variáveis de sistema para um programa operar a partir do endereço &HB000 ou para voltar a área normal de programas BASIC.

Os valores das variáveis de sistema de uma ou outra área de programa BASIC são armazenados a partir do endereço &HFFCA que é uma área não usada pelo sistema. Dependendo de qual área está sendo usada para operar um programa BASIC, os valores que apontam para lugares estratégicos do outro programa são temporariamente armazenados a partir daquele endereço.

É isso que possibilita que você possa operar um ou outro programa em regiões diferentes.

CHAMANDO COMANDOS DO BASIC

A maior parte do Interpretador BASIC ocupa a página 1 do SLOT 0, mas podem existir duas ou mais versões. O que realmente importa é que ele interprete rigorosamente o BASIC MSX, sem levar em conta o local da memória onde se encontra tal rotina ou se ela difere de um ou mais bytes de um outro MSX.

Mesmo sendo o Interpretador uma espécie de poço de areia movediça (é preciso tomar cuidado onde se está pisando), existe uma maneira de se simular os comandos do BASIC.

ENTRY POINTS DE COMANDOS

Para que o Interpretador possa localizar rapidamente os entry points (pontos de entrada) de um comando do BASIC, existe uma tabela que tem início no endereço &H392E (página 0). A organização dos entry points na tabela é feita em ordem crescente de acordo com o código da TOKEN.

A menor TOKEN de comando é a 129 (END). O primeiro entry point da tabela (2 bytes) corresponde a essa token, e assim sucessivamente até o final da tabela no endereço &H3B64.

Disso podemos tirar a fórmula:

$$EE = (CT - 129) * 2 + \&H392E$$

Onde:

EE = endereço de armazenamento do entry point
CT = código da tabela de comando

Por exemplo, suponhamos que você queira des-

cobrir o endereço onde se inicia a rotina do comando PLAY. O valor de sua token é 193. Usando esse código na fórmula obtemos o seguinte endereço:

&H39AE

Consultando o conteúdo desse endereço e do byte seguinte, pois o entry point está armazenado na forma LSB, MSB, obtemos o seguinte endereço:

&H73E5

Esse é o entry point do PLAY.

Ao chamar um comando do BASIC, você passará a operar literalmente em BASIC, o que aumenta a lentidão do seu programa.

Alguns comandos são desnecessários de serem chamados em um programa Assembly como: GOTO, SCREEN (as rotinas dos BIOS fazem isso com maior rapidez), REM, PUTSPRITE (é simples manipular sprites diretamente em Assembly) e outros comandos mais.

Um comando do BASIC deve ser chamado quando queremos que o programa faça algo que necessite um algoritmo muito complexo ou trabalhoso, como o comando CIRCLE ou o PLAY, por exemplo.

Vamos apresentar uma continuação (fig 1.22) para o programa TOKEN (fig 1.4). Após obter a token de um comando, ele imprimirá o seu entry point.

FIGURA 1.22 - Rotina End Exec

		;ROTINA ENDEEXEC.	
00A2		CHPUT: EQU 0A2H	
		ORG 0C05EH	
C05E 1A		LD A,(DE)	;IMPLEMENTA DESVIO
C05F CD00C1		CALL 0C100H	;P/ ROTINA EM
			;6HC100.
			;A CONTEM CODIGO
			DA TOKEN
		ORG 0C100H	
C100 06B1		LD B,B1H	;SUBTRAI 129.
C102 90		SUB B	
C103 87		ADD A,A	;MULTIPLICA POR 2.
C104 1600		LD D,0	;TRANSFERE PARA DE.
C106 5F		LD E,A	
C107 212E39		LD HL, 392EH	;SOMA COM ENDEREÇO
C10A 19		ADD HL,DE	;INICIAL DA TABELA
			;DE ENTRY POINTS
C10B 23		INC HL	;APONTA HL PARA
			;A PARTE ALTA.
C10C 0602		LD B,2	;LOOP P/ LER ENDEREÇO.
C10E 3E20		LD A,20H	;IMPRIME ESPACO.
C110 CDA200		CALL CHPUT	
C113	LOOP:		
C113 7E		LD A,(HL)	;ROTINA PARA
C114 E6F0		AND 11110000B	;IMPRESSAO DO
C116 1F		RRA	;NUMERO EM
C117 1F		RRA	;HEXADECIMAL.
C118 1F		RRA	
C119 1F		RRA	
C11A C630		ADD A,30H	
C11C FE3A		CP 3AH	

C11E 3802	JR	C,CONT	
C120 C607	ADD	A,7	
C122	CONT:		
C122 CDA200	CALL	CHPUT	
C125 7E	LD	A,(HL)	
C126 E60F	AND	00001111B	
C128 C630	ADD	A,30H	
C12A FE3A	CP	3AH	
C12C 3802	JR	C,CONT2	
C12E C607	ADD	A,7	
C130	CONT2:		
C130 CDA200	CALL	CHPUT	
C133 2B	DEC	HL	!APONTA HL PARA
			!A PARTE BAIXA.
C134 10DD	DJNZ	LOOP	!EFETUA LOOP.
C136 C9	RET		!FIM DA ROTINA.
C137	END		

Vejamos agora alguns exemplos de como utilizar algumas rotinas úteis do interpretador.

CIRCLE

FIGURA 1.23 - Programa CIRCLE

```

;PROGRAMA CIRCLE.
      ORG 0C000H
5B11  CIRCLE:EQU 5B11H
0072  INIGRP:EQU 0072H
009F  CHGET: EQU 009FH
C000  CD7200      CALL INIGRP      ;INICIALIZA SCREEN 2.
C003  210DC0      LD HL,LBASIC     ;HL DEVE APONTAR PARA
                                          ;UMA FALSA LINHA BASIC
C006  CD115B      CALL CIRCLE
C009  CD9F00      CALL CHGET      ;CHAMA CIRCLE.
                                          ;ESPERA UMA TECLA
C00C  C9          RET              ;SER PRESSIONADA
C00D  28313238 LBASIC:DEFB '(128,96),30',0 ;VOLTA AO BASIC.
C011  2C393629
C015  2C333000
C019                      END

```

Note que, quando simulamos o CIRCLE, o par HL deve apontar para uma tabela que contenha os códigos ASCII com a sintaxe do comando. Os números podem estar em ASCII ou na forma tokenizada.

DRAW

FIGURA 1.24 - Programa DRAW

```

;PROGRAMA DRAW.
      ORG 0C000H
5D6E  DRAW: EQU 5D6EH
0072  INIGRP:EQU 0072H
009F  CHGET: EQU 009FH
C000  CD7200      CALL INIGRP      ;INICIALIZA SCREEN2.
C003  210DC0      LD HL,STRING     ;HL DEVE APONTAR P/
                                          ;UMA FALSA STRING
C006  CD6E5D      CALL DRAW        ;CHAMA DRAW.
C009  CD9F00      CALL CHGET      ;ESPERA UMA TECLA
                                          ;SER PRESSIONADA
C00C  C9          RET              ;RETORNA AO BASIC.
C00D  STRING:
C00D  22533842      DEFB '"S8BM80,80NDUNR3U2E"
C011  4D38362C
C015  38304144
C019  554E5433
C01D  553245

```

```

C020 4E484634      DEFB 'NHF4BR2NU5BR3NE'
C024 4252324E
C028 55354252
C02C 334E45
C02F 4C48554E      DEFB 'LHUNR2UERFBR8DBFND2'
C033 52325545
C037 52464252
C03B 42444246
C03F 4E4432
C042 55455246      DEFB 'UERFDGLHBR4BDUNU3ERFD',0
C046 44474C48
C04A 42523442
C04E 44554E55
C052 33455246
C056 442200
C059               END

```

Para simular o comando DRAW basta apontar o par HL para uma string com os subcomandos do DRAW, exatamente como numa linha BASIC.

PLAY

FIGURA 1.25 - Programa PLAY

```

;PROGRAMA PLAY.
      ORG 0C000H
73E5  PLAY: EQU 73E5H
C000  LD HL,STRING ;HL DEVE APONTAR
                        ;P/ UMA FALSA
                        ;STRING
                        ;CHAMA PLAY.
                        ;RETORNA AO BASIC.
C003  CDE573      CALL PLAY
C004  C9          RET
C007  224F334C  STRING:DEFB "'03L2CE-GF'",
C00B  3243452D
C00F  4746222C
C013  224F334C  CONT: DEFB "'03L4CE-G04CL803B04CDE-FE-DC'",0
C017  3443452D
C01B  474F3443
C01F  4C384F33
C023  424F3443
C027  44452D46
C02B  452D4443
C02F  2200
C031               END

```

O modo de armazenamento do play é igual ao do comando DRAW, variando apenas os macrocomandos e o entry point.

PAINT

Para demonstrarmos o uso do PAINT, executaremos primeiramente um CIRCLE tal qual a figura 1.23 e em seguida vamos pintá-lo.

FIGURA 1.26 - Programa PAINT

```

;PROGRAMA PAINT.
      ORG 0B000H
5B11  CIRCLE:EQU 5B11H
59C5  PAINT: EQU 59C5H
0072  INIGRP:EQU 0072H
009F  CHGET: EQU 009FH

```

```

8000 CD7200      CALL INIGRP      ;INICIALIZA SCREEN 2.
8003 2113B0      LD   HL,LBASIC   ;HL DEVE APONTAR PARA
                                ;UMA FALSA LINHA BASIC
8006 CD115B      CALL CIRCLE      ;CHAMA CIRCLE.
8009 211FB0      LD   HL,FALINH   ;HL DEVE APONTAR PARA
                                ;UMA FALSA LINHA BASIC.
800C CDC539      CALL PAINT       ;CHAMA CIRCLE.
800F CD9F00      CALL CHGET       ;ESPERA UMA TECLA
                                ; SER PRESSIONADA
8012 C9          RET              ;VOLTA AO BASIC.
8013 28313238    LBASIC=DEFB '(128,96),30',0
8017 2C393629
801B 2C333000
801F 28313238    FALINH=DEFB '(128,96),15',0
8023 2C393629
8027 2C313500
802B             END

```

LINE

O comando LINE apresenta uma peculiaridade em relação aos comandos vistos até agora. O traço que liga as coordenadas dos dois vértices não deve ser o sinal se menos, mas sim a sua respectiva token (&HF2).

FIGURA 1.27 - Programa LINE

```

;PROGRAMA LINE.
4B0E             ORG   0C000H
0072             LINE: EQU 4B0EH
009F             INIGRP: EQU 0072H
C000 CD7200      CHGET: EQU 009FH
                                CALL INIGRP      ;INICIALIZA SCREEN 2.
C003 210DC0      LD   HL,LBASIC   ;HL DEVE APONTAR
                                ;PARA UMA FALSA
                                ;LINHA BASIC.
C006 CD0E4B      CALL LINE
C009 CD9F00      CALL CHGET       ;ESPERA UMA TECLA
                                ;SER PRESSIONADA.
C00C C9          RET              ;VOLTA AO BASIC.
C00D 2833302C    LBASIC=DEFB '(30,128)',0F2H,'(128,96)',0
C011 31323829
C015 F2283132
C019 3B2C3936
C01D 2900
C01F             END

```

Como você pode notar, a simulação de um determinado comando do BASIC é relativamente fácil. Na grande maioria dos comandos, basta apontar o par HL para uma falsa linha BASIC e chamar a devida rotina.

UTILIZANDO VARIÁVEIS EM COMANDOS

Quando chamamos comandos do BASIC, passamos literalmente a operar em BASIC. Então por que não tirarmos algum proveito disso? Afinal, devido à complexidade das rotinas do Interpretador, perdemos um tempo considerável de processamento.

Os comandos do BASIC permitem que usemos parâmetros numéricos e alfanuméricos.

3

As variáveis do sistema que devem ser alteradas são:

```
VARTAB (&HF6C2)
ARYTAB (&HF6C4)
STREND (&HF6C6)
```

Analisemos o programa da figura 1.28 .

FIGURA 1.28 - Programa CIRCON

```

;CIRCLE CONCENTRICO.
      ORG      0C0000H
F6C2  VARTAB: EQU  0F6C2H
F6C4  ARYTAB: EQU  0F6C4H
F6C6  STREND: EQU  0F6C6H
0072  INIGRP: EQU  0072H
009F  CHGET:  EQU  09FH
5B11  CIRCLE: EQU  5B11H

C000  2AC2F6      LD      HL, (VARTAB)      ;ARMAZENAR CONTEU-
C003  2261C0      LD      HL, (AVAR),HL      ;DO DAS VARIÁVEIS DE
C006  2AC6F6      LD      HL, (STREND)      ;SISTEMA QUE DELI-
C009  2263C0      LD      HL, (ASTR),HL      ;MITAR A ÁREA DE
C00C  2AC4F6      LD      HL, (ARYTAB)      ;VARIÁVEIS DO
C00F  2265C0      LD      HL, (ARYT),HL      ;BASIC.
C012  2158C0      LD      HL, NVAR
C015  22C2F6      LD      HL, (VARTAB),HL
C018  2160C0      LD      HL, FIMVAR
C01B  22C6F6      LD      HL, (STREND),HL
C01E  22C4F6      LD      HL, (ARYTAB),HL
C021  CD7200      CALL  INIGRP              ;ATIVA SCREEN2.
C024  060A      LD      B,10
C026
C026  2A5EC0      LD      HL, (NVAR+3)      ;CARREGA VALOR DA
;VARIÁVEL EM HL
C029  23
C02A  23      INC      HL
C02B  23      INC      HL
C02C  225EC0      LD      HL, (NVAR+3),HL    ;ARMAZENAR NOVO VALOR.
C02F  214FC0      LD      HL, LBASIC        ;APONTAR HL P/ LINHA
;BASIC.
;SALVAR B.
C032  C5      PUSH  BC
C035  CD115B      CALL  CIRCLE
C036  C1      POP   BC
C037  10ED      DJNZ  LOOP
C039  CD9F00      CALL  CHGET
;RECUPERA B.
;EFETUAR LOOP.
;ESPERAR TECLA SER
;PRESSIONADA

```

```

C03C 2A61C0      LD      HL,(AVAR)      ;RESTITUI O
C03F 22C2F6      LD      (VARTAB),HL    ;VALOR DAS
C042 2A63C0      LD      HL,(ASTR)      ;VARIABLES
C045 22C6F6      LD      (STREND),HL    ;DE SISTEMA.
C048 2A65C0      LD      HL,(ARYT)
C04B 22C4F6      LD      (ARYTAB),HL
C04E C9          RET
C04F 2B313238     LBASIC:DEFB ' (128,96),LZ ',0
C053 2C393629
C057 2C4C2500
C05B 024C0000     NVAR:  DEFB 02,'L',0,0,0
C05F 00
C060 00          FIHVAR:DEFB 0
C061 0000         AVAR:  DEFW 0
C063 0000         ASTR:  DEFW 0
C065 0000         ARYT:  DEFW 0
C067             END

```

ENTRY POINTS DE FUNÇÕES

Logo após a tabela de comandos, começa a tabela que contém os entry points das funções do BASIC. Seu endereço inicial é o &H39DE, e sua organização é igual à tabela de entry points de comandos.

Podemos obter facilmente o entrepoint utilizando a fórmula:

$$EE = (TF - 1) * 2 + \&H39DE$$

Onde:

EE= endereço de armazenamento do entry point
 TF= código da token de função (com o bit 7 resetado.)

A tabela da figura 1.29 mostra o entry point de cada uma das funções do BASIC.

FIGURA 1.29 - Tabela de entry points de funções

&H6861 LEFT\$	&H4FCC POS	&H30BE FIX
&H6891 RIGHT\$	&H67FF LEN	&H7940 STICK
&H689A MID\$	&H6604 STR\$	&H794C STRIG
&H2E97 SGN	&H68BB VAL	&H795A PDL
&H30CF INT	&H680B ASC	&H7969 PAD
&H2E82 ABS	&H681B CHR\$	&H7C39 DSKF
&H2AFF SQR	&H541C PEEK	&H6D39 FPOS
&H2BDF RND	&H7BF5 VPEEK	&H7C66 CVI
&H29AC SIN	&H6848 SPACE\$	&H7C6B CVS
&H2A72 LOG	&H65F5 OCT\$	&H7C70 CVD
&H2B4A EXP	&H65FA HEX\$	&H6D25 EOF
&H2993 COS	&H4FC7 LPOS	&H6D03 LOC
&H29FB TAN	&H65FF BIN\$	&H6D14 LOF
&H2A14 ATN	&H2F8A CINT	&H7C57 MKI\$
&H69F2 FRE	&H2FB2 CSNG	&H7C5C MKS\$
&H4001 INP	&H303A CDBL	&H7C61 MKD\$

OS HOOKS (DESVIOS)

Baseada na filosofia de modularidade e expansibilidade do padrão MSX, a Microsoft (criadora do Basic MSX) dotou o sistema operacional de um recurso extremamente poderoso: a possibilidade do usuário alterar o funcionamento das principais rotinas do BIOS e de alguns comandos do Basic sem a necessidade de adaptação do hardware ou de reprogramação da ROM.

Esta característica é possível graças à estrutura do sistema operacional que, antes de executar os comandos alteráveis, verifica em um setor especial da RAM se deve prosseguir normalmente ou tomar um caminho alternativo definido pelo usuário. Com isso você pode alterar as características de entrada e saída do computador, criar novos comandos e até conectar periféricos não-MSX ao micro, mantendo a compatibilidade entre os dois.

O QUE SÃO OS HOOKS

Basicamente, os "hooks" comportam-se como um desvio de trem: quando um comando chama um "hook" este nada faz, devolvendo o controle à ROM. Mas o usuário pode, com auxílio de POKes, alterar a "direção" do comando, ou seja, desviá-lo.

Nem todos os comandos e funções do Basic e rotinas do BIOS possuem "hooks", mas apenas aqueles em que os "hooks" podem ter alguma utilidade, como veremos mais a frente.

COMO OS HOOKS FUNCIONAM

Um "hook" é uma sequência de cinco bytes. Existem diversos "hooks" enfileirados na área final da página 3 da RAM, mais exatamente entre os endereços &HFD9A e &HFFCA, que são preenchidos com o código &HC9 (RET) ao se ligar o computador.

Em cada "hook", cinco bytes estão reservados para cada comando ou função. Quando um comando é chamado, ele executa um CALL para o primeiro byte do "hook" que, possuindo o byte &HC9 (RET), faz o mesmo ser ignorado pelo sistema.

Para causar o "desvio" citado acima, deve-se fazer uma sequência de até cinco POKes na área correspondente ao respectivo comando.

Vejamos um exemplo prático: vamos alterar o computador para que um "bip" seja tocado ao se pressionar uma tecla. Para isso deve-se saber, antes de mais nada, se existe um "hook" para a rotina de teclado.

Pesquisando na tabela de "hooks", constatamos que existe um hook para a rotina CHGET do BIOS, responsável pela leitura do teclado. Esse "hook" inicia em &HFDC2 e termina em &HFDC6. Existe também no BIOS uma rotina chamada BEEP (&H00C0), que é a utilizada normalmente pelo micro.

Experimente comandar:

```
POKE &HFDC2,&HC3:POKE &HFDC3,&HC0:POKE &HFDC4,&H00
```

Com isso o "hook" ficará assim:

```
C3 C0 00 C9 C9
```

Depois, veja os resultados (pressione algumas teclas).

Agora faça o seguinte: desligue o computador e ligue novamente (RESET total), e então digite:

```
POKE &HFDC2,&HC3 (RETURN)  
POKE &HFDC3,&HC0 (RETURN)  
POKE &HFDC4,&H00 (RETURN)
```

Se você procedeu corretamente, verá que perdeu o controle do sistema logo após digitar a primeira linha. Por que isso ocorre?

Pensando-se um pouco, vem a resposta: após o primeiro POKE, o "hook" ficou assim:

```
C3 C9 C9 C9 C9
```

Ao pressionarmos novamente uma tecla, o computador imediatamente executou o primeiro comando do "hook", que é:

```
JP C9C9 (C3 C9 C9)
```

Como é extremamente improvável que exista algo lógico em &HC9C9 e subsequentes, o micro só pode mesmo entrar em "crash". Por isso, tome cuidado ao alterar os "hooks", pois alguns são chamados em instantes críticos como, por exemplo, sempre que alguma tecla é pressionada.

ALGUMAS APLICAÇÕES

Existem vários "hooks" que podem ser úteis para o usuário. Por exemplo, imagine que você queira proteger um programa em BASIC, impedindo que este pos-

sa ser listado. Para bloquear o comando LIST, basta alterar seu "hook" (iniciado em &HFFB9) para um outro comando. Digite um programa qualquer em BASIC e comande:

```
POKE &HFFB9,&HC1
LIST
```

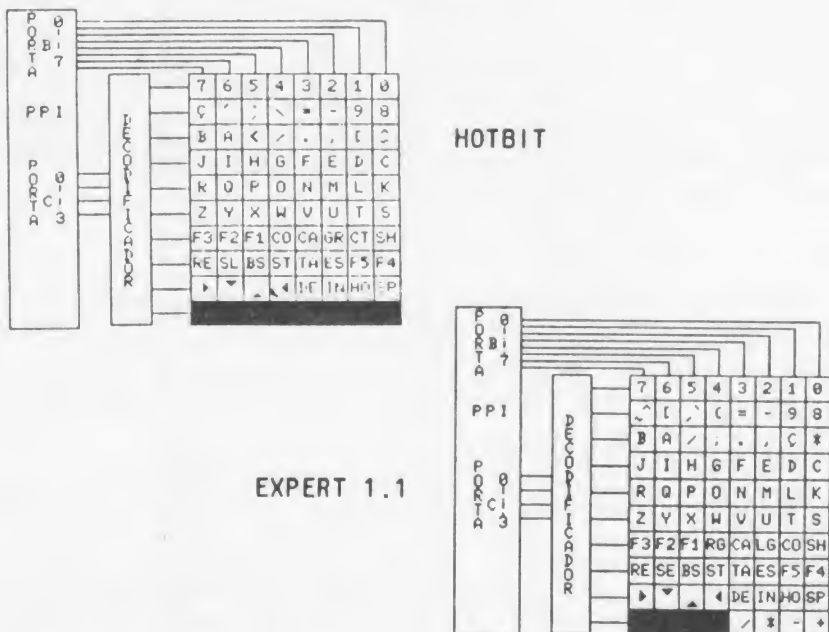
Aparentemente, não há nenhum programa na memória. Se você digitar o comando RUN, entretanto, verá que o programa continua funcionando! Você pode também ser mais "cruel" e alterar o "hook" do LIST para que este cause uma pane geral no computador ao se tentar listar o programa.

No Apêndice II, você tem todos os "hooks" do computador listados.

A MATRIZ DO TECLADO

O circuito que controla o gerenciamento do teclado é a PPI. O teclado do MSX apresenta uma organização matricial conforme pode ser observado na figura 1.30.

FIGURA 1.30 - Configurações de teclado



O sistema operacional faz a varredura do teclado, permanentemente, linha após linha, enviando para o primeiro nibble da porta C da PPI o número da linha da matriz do teclado a ser ativada.

A PPI envia esse número ao decodificador do teclado, que ativa a linha da matriz.

Em seguida, a porta B da PPI é consultada. Todas as colunas que possuírem um bit ressetado indicarão o pressionamento de uma tecla.

Após o sistema operacional receber esse byte ele consulta uma das 5 tabelas (de 48 bytes cada) da ROM que estão a partir do endereço &H0DA5, para saber o código do caractere correspondente à tecla pressionada.

As primeiras 5 tabelas são para as primeiras 5 linhas da matriz, que correspondem às teclas que são de controle ou especiais.

O Sistema Operacional seleciona a tabela de acordo com as teclas que estiverem sendo pressionadas conjuntamente (SHIFT, teclas gráficas, SHIFT+teclas gráficas).

A tabela que indica o código das linhas de 6 a 10 do teclado tem início a partir do endereço &H1033. Essa tabela não faz parte das outras, porque o código das teclas é o mesmo se outras teclas estiverem sendo pressionadas conjuntamente.

FIGURA 1.31 - Tabelas de reconhecimento de teclas especiais

1033	00	00	00	00	00	00	00	00	00	Linha 6
103B	00	00	1B	09	00	08	1B	0D		Linha 7
1043	20	0C	12	7F	1D	1E	1F	1C		Linha 8
104B	2B	2D	2A	2F	00	00	00	00		Linha 9
1053	00	00	00	00	00	00	00	00	00	Linha 10
Coluna	1	2	3	4	5	6	7	8		

O BUFFER DO TECLADO

Após ser feita a varredura do teclado e o código do caractere ter sido obtido, ele é enviado para uma região da memória denominada "buffer" do teclado.

Tecnicamente, o buffer de teclado do MSX é chamado de KEYBUF e situa-se entre os endereços &HFBF0 e &HFC17 (inclusive), num total de 40 bytes. O funcionamento do buffer é igual ao de um banco: se o caixa trabalhar rápido, quem chega é atendido na hora e não há possibilidade de formar fila. Isso é o que acontece, por exemplo, quando você digita um programa em BASIC.

Se o caixa trabalhar devagar (ou parar), logo teremos a formação de uma fila que cresce com o tempo até chegar à calçada! Nessa hora quem chegar vai desistir ao ver o tamanho da fila. Isso é o que chamamos de "buffer lotado", e acontece quando o computador está executando uma tarefa muito demorada e você pressiona várias teclas até o click cessar. Então, quando o ele acaba a execução ficando livre, tudo aquilo que foi pressionado aparece na tela e é processado, como o caixa que voltou a funcionar tão rápido que esvaziou a fila!

INSERINDO DADOS NO KEYBUF

A possibilidade de colocar caracteres no buffer do teclado pode ser aproveitada para simular sua digitação, ou seja, fazer o computador "pensar" que algo foi digitado. Isso parece inútil a princípio, mas para um bom programador toda informação pode ser útil, especialmente na arte de proteger e desproteger programas.

Além dos 40 bytes, existem duas variáveis do sistema (PUTPNT, no endereço &HF3FB, e GETPNT, no endereço &HF3FA) que controlam a entrada e saída dos caracteres do buffer.

PUTPNT aponta o byte livre para o próximo carácter digitado, e GETPNT aponta para o próximo carácter a ser retirado do buffer. A cada novo carácter digitado, PUTPNT avança de um endereço, voltando ao começo sempre que tenta escapar do buffer.

GETPNT é incrementada sempre que um carácter é retirado do buffer pelo sistema operacional, também voltando ao começo. O buffer é dado como cheio quando PUTPNT tenta apontar para o mesmo endereço dado por GETPNT.

Para colocar uma informação no buffer deve-se, além de escrevê-la no espaço reservado, acertar as variáveis PUTPNT e GETPNT. Por exemplo, para inserir o comando

RUN (RETURN)

deve-se alterar os bytes da forma como está especificado na figura 1.32.

FIGURA 1.32 - Inserindo um comando no KEYBUF

```
KEYBUF:  FBF0 52  'R'
         FBF1 55  'U'
         FBF2 4F  'N'
         FBF3 0D  RETURN

PUTPNT:  F3F8 F4   ;Último endereço + 1
         F3F9 FB
GETPNT:  F3FA F0   ;Primeiro endereço
         F3FB FB
```

Estes podem ser colocados por um programa Assembly ou por um programa em BASIC (comando POKE). Ao retornar ao nível de comando direto, o computador executará a ordem contida no buffer como se esta tivesse sido digitada. Mas atenção: os comandos e funções do BASIC que lêem o teclado como o INPUT, LINEINPUT, INPUT\$, INKEY\$, etc reconhecerão os dados no buffer como teclas pressionadas e os retirarão do mesmo. O mesmo vale para a rotina CHGET do BIOS.

Teste o programa da figura 1.33 .

FIGURA 1.33 - Inserindo comandos em BASIC no KEYBUF

```
10 SCREEN 1
20 A$="COLOR 15,4,7"+CHR$(13)
30 FOR I=1 TO LEN(A$)
40 POKE I+&HFBF0,ASC(MID$(A$,I,1))
50 NEXT I: POKE &HF3FA,&HF0:POKE &HF3FB,&HFB
60 POKE &HF3F8,&HFD:POKE &HF3F9,&HFB
70 END
```

O programa coloca no buffer a ordem para alterar as cores da tela. Esse mesmo programa poderia ser facilmente feito em Assembly, como mostra a figura 1.34 .

FIGURA 1.34 - Programa KEYBUF

```
F3FB      PUTPNT:EQU  :           0F3FBH
F3FA      GETPNT:EQU  :           0F3FAH
FBF0      KEYBUF:EQU  :           0FBF0H

          ORG      0C000H
C000 2119C0      LD      HL,COMANDO      ;TRANSFERE COMANDO
C003 11F0FB      LD      DE,KEYBUF      ;PARA O BUFFER
C006 010D00      LD      BC,13          ;DO TECLADO
C009 ED80        LDIR
C00B 21F0FB      LD      HL,KEYBUF      ;APONTA GETPNT P/
C00E 22FAF3      LD      (GETPNT),HL    ;INICIO DO COMANDO
C011 110E00      LD      DE, 14         ;APONTA PUTPNT P/
C014 19          ADD     HL,DE          ;O FIM DO COMANDO
C015 22F8F3      LD      (PUTPNT),HL    ;
C018 C9          RET                   ;RETORNA AO BASIC
C019 434F4C4F COMANDODEFB 'COLOR 15,4,7',13
C01D 52203135
```

C021 2C342C37
C025 0D
C026

END





OPERANDO O VDP EM LINGUAGEM DE MÁQUINA

O sistema MSX é provido de rotinas específicas para o controle e inicialização do VDP e da VRAM. Explicaremos aqui como tirar proveito delas (no apêndice V há uma lista completa com todas as rotinas do BIOS).

As primeiras rotinas a comentar são DISSCR (&H41) e ENASCR (&H44) que, quando chamadas, chaveiam a imagem do VDP. DISSCR faz a imagem desaparecer, ficando a tela toda com a cor do fundo (quando em SCREEN 0) ou da borda (nas demais telas). ENASCR inverte o processo.

Vejamos um programa exemplo na figura 2.1

FIGURA 2.1 - Programa BACKCOLOR.

0041	ORG	0C000H	
0044	DISSCR=EQU	0041H	
F3EA	ENASCR=EQU	0044H	
009F	BDRCLR=EQU	0F3EAH	
0062	CHGET=EQU	009FH	
C000	CHGCLR=EQU	0062H	
C003	CALL	DISSCR	;DESABILITA VIDEO.
C003	INIC:		
C006	CALL	CHGET	;ESPERA TECLA SER
C008	CP	2B	;PRESSIONADA.
C00A	JR	NZ,CONT	;CASO SEJA 'P',
C00D	LD	A, (BDRCLR)	;PEGA COR DE FUN-
C00E	INC	A	;DO E INCREMENTA.
C011	LD	(BDRCLR),A	;ARMAZENA EM BDRCLR.
C013	JR	CONT1	;SALTA P/ CONT1.
	CONT:		

C013 FE1D		CP 29	;CASO "4",
C015 200C		JR NZ,FIN	
C017 3AEAF3		LD A,(BDRCLR)	;PEGA COR DE FUNDO
C01A 3D		DEC A	;E DECREMENTA.
C01B 32EAF3		LD (BDRCLR),A	;ARMAZENA EM BDRCLR.
C01E	CONT1:		
C01E CD6200		CALL CHGCLR	;MUDA COR DE FUNDO.
C021 18E0		JR INIC	;SALTA P/ INIC.
C023	FIN:		
C023 CD4400		CALL ENASCR	;HABILITA VIDEO.
C024 C9		RET	;RETORNA AO BASIC.
C027		END	

O programa BACKCOLOR utiliza a rotina DISSCR para interromper a geração da imagem pelo VDP. Em seguida é chamada a rotina CHGET (&H009F), que espera o pressionamento de uma tecla e devolve o seu código. Se a tecla for a seta para a direita a variável fundo é incrementada e enviada para o registo 7 do VDP que determina a cor de fundo.

Para enviar um valor a um registo do VDP usamos a rotina WRTVDP (&H0047) que possui um funcionamento bastante simples, basta carregar o registrador B com o valor a ser enviado para o registo, e no registrador C o número do registo do VDP.

Se for pressionada a tecla de seta para a esquerda, a variável fundo é decrementada, sendo seu valor enviado ao registo 7 do VDP.

Se a tecla de seta para cima for pressionada, o programa habilita novamente o VDP pela rotina ENASCR e retorna ao BASIC.

LENDO E ESCRREVENDO NA VRAM

Outro importante par de rotinas é RDVRM(&H4A) e WRTVRM(&H4D) que manipula as posições de memória da VRAM. RDVRM lê o conteúdo do endereço especificado em HL e o devolve em A. Reciprocamente, WRTVRM escreve no endereço da VRAM apontado por HL o conteúdo dado por A.

No programa da figura 2.2 vamos utilizar as rotinas RDVRM e WRTVRM para fazermos um DUMP da VRAM na impressora. O byte que estiver sendo lido e impresso ficará temporariamente com o código &HFF.

FIGURA 2.2 - Programa VDUMP

	ORG 0C000H	
004D WRTVRM= EQU	004DH	
004A RDVRM= EQU	004AH	
009F CHGET= EQU	009FH	
00A5 LPTOUT= EQU	00A5H	
C000 210000	LD HL,0	;POSICIONA CURSOR

C003	INIC:			!NO INICIO DA RAM
C003 54		LD	D,H	!
C004 5D		LD	E,L	!CARREGA DE COM 0
C005 CD4A00		CALL	RDURM	!END. DO CURSOR.
C008 3278C0		LD	(VALOR),A	!ARMAZENA VALOR
C008 3EFF		LD	A,0FFH	!SOB O CURSOR.
C00D CD4D00		CALL	WRTURM	!IMPRIME CURSOR.
C010 CD9F00		CALL	CHGET	!
C013 FE50		CP	'P'	!ESPERA COMANDO.
C015 202C		JR	NZ,CONT	!COMPARA COM 'P'.
C017 3A78C0		LD	A,(VALOR)	!SE <> SALTA P/CONT.
C01A E6F0		AND	11110000B	!IMPRIME 0
C01C 1F		RRA		!HEXADECIMAL
C01D 1F		RRA		!DO BYTE SOB
C01E 1F		RRA		!O CURSOR.
C01F 1F		RRA		
C020 C630		ADD	A,30H	
C022 FE3A		CP	3AH	
C024 3802		JR	C,IMP	
C026 C607		ADD	A,7	
C028	IMP:			
C028 CDA500		CALL	LPTOUT	
C028 3A78C0		LD	A,(VALOR)	
C02E E60F		AND	00001111B	
C030 C630		ADD	A,30H	
C032 FE3A		CP	3AH	
C034 3802		JR	C,IMP1	
C036 C607		ADD	A,7	
C038	IMP1:			
C038 CDA500		CALL	LPTOUT	
C038 3E20		LD	A,20H	
C03D CDA500		CALL	LPTOUT	
C040 23		INC	HL	
C041 181F		JR	CONT4	
C043	CONT:			
C043 FE46		CP	'F'	!SE 'F' FOI PRES-
C045 C8		RET	Z	!SIONADO, VOLTA
				!AO BASIC.
C046 FE1C		CP	28	!SE 'D' FOR
C048 2001		JR	NZ,CONT1	!PRESSIONADA
C04A 23		INC	HL	!AVANCA O CURSOR.
C048	CONT1:			
C048 FE1D		CP	29	!SE '4' FOR PRES-
C04D 2001		JR	NZ,CONT2	!SIONADA RETROCEDE
C04F 28		DEC	HL	!O CURSOR.
C050	CONT2:			
C050 FE1E		CP	30	!SE 'A' FOR PRES-
C052 2805		JR	Z,CONT3	!SIONADA AVANCA
C054 060A		LD	B,10	!O CURSOR EM 10
C056	LOOP1:			!BYTES.
C056 23		INC	HL	
C057 10FD		DJNZ	LOOP1	
C059	CONT3:			
C059 FE1F		CP	31	!SE 'V' FOR PRES-
C05B 2805		JR	Z,CONT4	!SIONADA RETROCEDE
C05D 060A		LD	B,10	!O CURSOR EM 10
C05F	LOOP2:			!BYTES.
C05F 28		DEC	HL	
C060 10FD		DJNZ	LOOP2	
C062	CONT4:			
C062 FE0D		CP	0DH	!SE RETURN FOR
C064 2008		JR	NZ,CONT5	!PRESSIONADO
C066 CDA500		CALL	LPTOUT	!AVANCA O PAPEL
C069 3E0A		LD	A,0AH	!DA IMPRESSORA.
C06B CDA500		CALL	LPTOUT	
C06E	CONT5:			
C06E 3A78C0		LD	A,(VALOR)	!RESTITUI O BYTE
C071 EB		EX	DE,HL	!SOB O CURSOR.
C072 CD4D00		CALL	WRTURM	
C075 EB		EX	DE,HL	
C076 188B		JR	INIC	!EFETUA LOOP.
C07B 00	VALOR:	DEFB	0	
C079 0000	END:	DEFW	0	
C07B	END			

Quando o programa começa a operar, surge, dependendo da SCREEN em uso, um pseudo-cursor no canto esquerdo superior do vídeo. As teclas de controle do cursor também orientam a movimentação deste.

A tecla de seta para cima retrocede o cursor em 10 bytes, e a seta para baixo avança 10 bytes.

A tecla de seta para esquerda retrocede o cursor e a seta para direita avança o cursor.

Para se mandar um byte para a impressora, basta pressionar a tecla "P". Quando isto ocorre o cursor avança automaticamente.

Se, ao mandar imprimir um byte, ele não for impresso, não se preocupe. Provavelmente é porque a sua impressora possui um buffer, e só começará a imprimir quando forem enviados os caracteres &H0D e &H0A (Carriage Return e Line Feed). Para que esses caracteres sejam enviados para a impressora basta pressionar RETURN.

TROCANDO O MODO DAS TELAS

As rotinas INITXT (&H006C), INIT32 (&H006F), INIGRP (&H0072) e INIMLT (&H0075) equivalem ao comando SCREEN do BASIC. O seu uso é bastante simples, basta chamar a rotina com um CALL para que sejam organizadas todas as áreas da VRAM e registros do VDP.

O único cuidado que deve ser tomado é quanto ao valor dos registradores quando as rotinas de mudança de SCREEN são chamadas, pois elas destroem o valor de todos os registradores. Portanto antes de chamar essas rotinas guarde os valores importantes no stack para que não ocorram "crashes" na execução.

CONTROLANDO SPRITES EM LM

Os sprites são mapeados na VRAM em duas partes:

- 1) Tabela de atributos (iniciada em 6912).
- 2) Tabela de formação (iniciada em 14336).

Nesta seção, consideraremos sempre os endereços de "default", ou seja, com os valores dados pelo computador quando é ligado.

A tabela de atributos define os parâmetros dos 32 sprites do VDP e ocupa 128 bytes, agrupados de 4 em 4, como na figura 2.3. Essa tabela é sempre a mesma para as SCREENs 1, 2 e 3.

FIGURA 2.3 - Tabela de atributos dos SPRITES.

```

    SPRITE 0:6912 Y
              6913 X
              6914 N
              6915 C

    SPRITE 1:6916 Y
              6917 X
              6918 N
              6919 C
              "
              "
              "
    SPRITE 31:7036 Y
              7037 X
              7038 N
              7039 C
    
```

Onde :

Y:Coordenada Y do SPRITE.
 X:Coordenada X do SPRITE.
 N:Número do SPRITE.
 C:Cor do SPRITE.

A tabela de atributos é a mesma para definir sprites 8x8 e 16x16, com exceção do valor de N, que no segundo caso deve ser multiplicado por 4. Por exemplo, ao se comandar

```
PUTSPRITE 2,(160,40),15,3
```

teremos, como mostra a figura 2.4, na VRAM.

FIGURA 2.4 - Dump de parte da tabela de atributos.

MODO	8x8	16x16	
Endereço	6920 - 40	40	(Y)
	6921 - 160	160	(X)
	6922 - 3	12	(N)
	6923 - 15	15	(C)

A tabela de formação dos sprites inicia em 14336 (para 8x8 e 16x16).No sistema 8x8 cada grupo de 8 bytes representa um sprite completo, e no sistema 16x16 cada imagem é definida por 32 bytes. Note que o espaço ocupado por um sprite 16x16 é quatro vezes maior que o ocupado por um 8x8.

A estrutura dessa tabela é apresentada na figura 2.5

FIGURA 2.5 - Tabela de formação dos sprites

Número	End. (8x8)	end. (16x16)
0	14336	14336
1	14344	14368
:	14352	14400
:	:	:
63	14480	16352
:	:	-----
:	:	-----
255	16376	-----

Não se esqueça de que o número pode variar de 0 a 255 (8x8) ou 0 a 63 (16x16). É fácil ver que o endereço inicial de uma figura (SPRITE) é dado por $E=14336+8*N$, onde N é o número definido na tabela de atributos. Isso explica o porquê de multiplicar N por 8 no sistema 16x16.

MOVIMENTANDO UM SPRITE

A esta hora você já deve estar pensando como comandar PUTSPRITE sem usar essa palavra do BASIC.

Você pode dar VPOKEs se estiver usando o interpretador, mas isto não pode ser feito diretamente em LM. Como para tudo existe uma solução, há uma rotina no BIOS chamada WRTVRM (&H004D) que funciona como um VPOKE: fornece-se o endereço em HL, o dado em A e chama-se &H004D, como se vê na figura 2.6 .

FIGURA 2.6 - Movimentando um SPRITE em Assembly.

ASSEMBLY				BASIC
XXXX	21	00	1B	VPOKE 6912,149
XXXX+3	3E	FF	LD A,149	
XXXX+5	CD	4D	00	
XXXX+8	C9		RET	

Assim, é possível não apenas definir um SPRITE na tela, mas também movimentá-lo. A velocidade conseguida por este processo é muito grande serve perfeitamente para a animação de jogos que usam SPRITES.

Vamos agora comprovar essa velocidade com um programa (fig 2.7) que define um sprite na tela e o movimenta da esquerda para a direita repetidas vezes. As setas de controle do cursor controlam a pausa, aumentando-a (seta para baixo) ou diminuindo-a (seta para cima).

FIGURA 2.7 - Movimentador de SPRITES

```

                                ORG    0C000H
0047      WRTVDP= EQU    047H
004A      RDVRM= EQU    04AH
004D      WRTVRM= EQU    04DH
005C      LDIRVM= EQU    05CH
006F      INIT32= EQU    06FH
0084      CALPAT= EQU    084H
0087      CALATR= EQU    087H
00D5      GTSTCK= EQU    0D5H
F3E0      RGISAV= EQU    0F3E0H

; INICIO

C000 CD6F00      CALL    INIT32      ; MODO SCREEN 1.
C003 3AE0F3      LD      A, (RGISAV) ; PASSA SPRITES
C006 CBC7        SET     0, A        ; PARA MODO AM-
C008 47          LD      B, A        ; PLIADO Bx8.
C009 0E01        LD      C, 1
C00B CD4700      CALL    WRTVDP
C00E 3E00        LD      A, 0
C010 CD8400      CALL    CALPAT      ; PROCURA INICIO
C013 EB          EX      DE, HL      ; DO PADRAO 0.
C014 217AC0      LD      HL, TBL      ; CRIA O PADRAO.
C017 010B00      LD      BC, 8
C01A CD5C00      CALL    LDIRVM
C01D 3E00        LD      A, 0
C01F CD8700      CALL    CALATR      ; BUSCA ATRI-
                                ; BUTOS DO
                                ; SPRITE 0.
                                ; COORD. Y.

C022 3E60        LD      A, 96
C024 CD4D00      CALL    WRTVRM
C027 23          INC     HL
C028 23          INC     HL
C029 3E00        LD      A, 0
C02B CD4D00      CALL    WRTVRM      ; PADRAO 0.
C02E 23          INC     HL
C02F 3E08        LD      A, 8
C031 CD4D00      CALL    WRTVRM      ; COR B (VER-
C034 3E20        LD      A, 32      ; MELHO).
C036 3279C0      LD      (TEMP), A   ; AJUSTA 0
C039             ; TEMPO.
C039             ; LOOP EXTERNO.
C039 0600        LD      B, 0        ; PREPARA 0
                                ; LOOP INTERNO.

C03B             ; LOOP:
C03B C5          PUSH    BC          ; SALVA BC.
C03C 3E00        LD      A, 0
C03E CD8700      CALL    CALATR      ; SPRITE 0.
C041 23          INC     HL          ; COORD. X.
C042 CD4A00      CALL    RDVRM      ; LE O VALOR.
C045 3C          INC     A           ; AVANCA 1.
C046 CD4D00      CALL    WRTVRM      ; ESCRIVE DE
                                ; VOLTA.
C049 3A79C0      LD      A, (TEMP)   ; MARCA O TEM-
C04C 47          LD      B, A        ; PO.
C04D             ;
C04D             ; TEMPO:
C04D 10FE        DJNZ    TEMPO
C04F C1          POP     BC          ; VOLTA BC.
C050 10E9        DJNZ    LOOP      ; FECHA LOOP.
C052 AF          XOR     A           ; LE SETAS DE
C053 CD0500      CALL    GTSTCK      ; CURSOR.
C056 FE01        CP      1           ; SOBE.
C058 2B09        JR      Z, RAPID
C05A FE05        CP      5           ; DESCE.
C05C 2B0F        JR      Z, LENTO
C05E FE03        CP      3           ; DIREITA.

```

```

C060 CB          RET Z
C061 1BD6        JR LOOP0      ;FECHA LOOP0.
C063            RAPID:
C063 3A79C0      LD A,(TEMP)    ;DECREMENTA A
C066 FE01        CP 1          ;VARIÁVEL TEMP.
C068 2BD1        JR Z,LOOP
C06A 3D          DEC A
C06B 1B07        JR CONT
C06D            LENTO:
C06D 3A79C0      LD A,(TEMP)    ;INCREMENTA A
C070 A7          AND A          ;VARIÁVEL TEMP.
C071 2BCB        JR Z,LOOP
C073 3C          INC A
C074            CONT:
C074 3279C0      LD (TEMP),A    ;ATUALIZA O VA-
C077 1BC2        JR LOOP       ;LOR DE TEMP.
C079            TEMP:
C07A FF          TBL:          DS 1
C07B 81          DB 0FFH       ;PADRÃO DO
C07C 81          DB 81H        ;SPRITE.
C07D 81          DB 81H
C07E 81          DB 81H
C07F 81          DB 81H
C080 81          DB 81H
C081 FF          DB 0FFH
C082            END

```

CHAVEANDO SPRITES

Vamos apresentar agora uma maneira de se fazer com que um sprite mude de forma rapidamente.

A idéia é muito simples. Basta definirmos vários padrões (logicamente na tabela de padrões) e quando quisermos mudar rapidamente a forma do sprite basta alterar na tabela de atributos o terceiro elemento que controla o sprite, apontando-o para outro padrão (outro SPRITE). A definição dos padrões pode ser feita através da rotina LDIRVM do BIOS, que permite que transportemos um bloco de bytes da RAM para a VRAM.

O programa da figura 2.8 irá controlar um SPRITE pela tela, conforme uma tecla de controle do cursor seja pressionada, indicando uma direção. Além de movimentar o sprite na direção indicada ele alterará, na tabela de atributos, o número do SPRITE fazendo com que o sprite mostrado tenha o seu padrão alterado rapidamente.

FIGURA 2.8 - Programa chavsprite.

```

                                ORG 0C000H
009F          CHGET: EQU 07FH
005C          LDIRVM:EQU 05CH
006F          INIT32:EQU 06FH
C000 CD6F00    CALL INIT32      ;TRANSFERE PARA A
C003 2165C0    LD HL,SPRITE     ;VRAM OS BYTES
C006 110030    LD DE,3800H      ;QUE DEFINEM OS
C009 018000    LD BC,128        ;4 SPRITES.
C00C CD5C00    CALL LDIRVM
C00F 1B12      JR TECLA
C011          INIC:

```

C011 F1	POP AF	RECUPERA TECLA
C012 D61C	SUB 28	PRESSIONADA
		SUBTRAI 28 PARA
		SELECIONAR SPRITE
		CORRETO
C014 3263C0	LD (DIREC),A	ARMAZENA NR DO PADRAO.
C017 2161C0	LD HL,CORY	TRANSFERE OS DADOS
C01A 11001B	LD DE,1800H	PARA A
C01D 010400	LD BC,4	TABELA DE
C020 CD5C00	CALL LDIRVM	ATRIBUTOS.
C023	TECLA:	ESPERA TECLA SER
C023 CD9F00	CALL CHGET	PRESSIONADA.
C026 F5	PUSH AF	ARMAZENA CODIGO DA
		TECLA NO STACK
C027 FE1C	CP 28	VERIFICA SE é '►'.
C029 200A	JR NZ,CONT	CASO AFIRMATIVO,
C02B 3A62C0	LD A,(CORY)	INCREMENTA
C02E C601	ADD A,1	COORDENADA X
C030 3262C0	LD (CORY),A	DO SPRITE.
C033 18DC	JR INIC	SALTA P/ INIC.
C035	CONT:	
C035 FE1D	CP 29	VERIFICA SE é '◄'.
C037 200A	JR NZ,CONT1	CASO AFIRMATIVO,
C039 3A62C0	LD A,(CORY)	DECREMENTA
C03C D601	SUB 1	COORDENADA X
C03E 3262C0	LD (CORY),A	DO SPRITE.
C041 18CE	JR INIC	SALTA P/ INIC.
C043	CONT1:	
C043 FE1E	CP 30	VERIFICA SE é '▼'.
C045 200A	JR NZ,CONT2	CASO AFIRMATIVO,
C047 3A61C0	LD A,(CORY)	DECREMENTA
C04A D601	SUB 1	COORDENADA Y
C04C 3261C0	LD (CORY),A	DO SPRITE.
C04F 18C0	JR INIC	SALTA P/ INIC.
C051	CONT2:	
C051 FE1F	CP 31	VERIFICA SE é '▲'.
C053 200A	JR NZ,CONT3	CASO AFIRMATIVO,
C055 3A61C0	LD A,(CORY)	INCREMENTA
C05B C601	ADD A,1	COORDENADA Y
C05A 3261C0	LD (CORY),A	DO SPRITE.
C05D 18B2	JR INIC	SALTA P/ INIC.
C05F	CONT3:	
C05F F1	POP AF	ILIMPA STACK.
C060 C9	RET	RETORNA AO BASIC.
C061 60	CORY: DEFB 96	
C062 80	CORY: DEFB 128	
C063 000F	DIREC: DEFB 0,15	
C065 0010181C	SPRITE: DEFB 00H,10H,18H,1CH	
C069 FE1C1810	DEFB 0FEH,1CH,18H,10H	
C06D 0008183B	SPRIT1: DEFB 00H,08H,18H,3BH	
C071 7F3B180B	DEFB 7FH,3BH,18H,0BH	
C075 00081C3E	SPRIT2: DEFB 00H,08H,1CH,3EH	
C079 7F08080B	DEFB 7FH,08H,08H,0BH	
C07D 0808087F	SPRIT3: DEFB 08H,08H,08H,7FH	
C081 3E1C0800	DEFB 3EH,1CH,08H,00H	
C085	END	

ORGANIZAÇÃO DA SCREEN 3

Por ocasião da publicação do "Aprofundando-se no MSX", dissemos que a SCREEN 3 era organizada de forma complexa e seu estudo detalhado não seria importante, já que o livro tratava quase sempre da linguagem BASIC. Agora que estamos falando de linguagem de máquina, achamos oportuno discutir como ela funciona.

A organização da VRAM quando o VDP está no modo SCREEN 3 é dada pela figura 2.9

FIGURA 2.9 - Tabelas que compõem a SCREEN 3

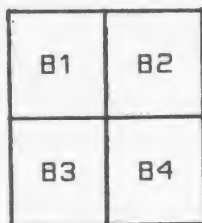
0-2047:	Tabela de padrões dos caracteres.
2048-2816:	Tabela dos códigos impressos na tela.
6912-7039:	Tabela de atributos dos sprites.
14336-16383:	Tabela de padrões dos sprites.

O funcionamento e as tabelas dos sprites são iguais às das outras SCREENs. O que vamos explicar são as duas primeiras tabelas.

A primeira tabela indica quais caracteres estão "impressos" na tela (à semelhança da SCREEN 2) e para facilitar o entendimento ela será comentada posteriormente, pois seu conteúdo é um pouco diferente.

Convém aqui explicar que na SCREEN 3 um caractere é diferente do normal: ele possui duas linhas de dois blocos (total=4), e cada bloco mede 4 pontos da SCREEN 2 (ver figura 2.10).

FIGURA 2.10 - Partes de um caractere da SCREEN 3.



Um bloco pode ter uma das 16 cores geradas pelo VDP. Os blocos B1 e B2 são representados por um mesmo byte (4 bits da esquerda para B1, 4 bits da direita para B2); B3 e B4 usam o byte seguinte (cada caractere exige dois bytes para ser definido).

A tabela de padrões define as cores de B1 e B4, e, portanto, cada caractere usa dois bytes. Ela comporta 1024 caracteres (0-255 quatro vezes) e é dividida em quatro partes (fig 2.11).

FIGURA 2.11 - Organização da tabela de padrões.

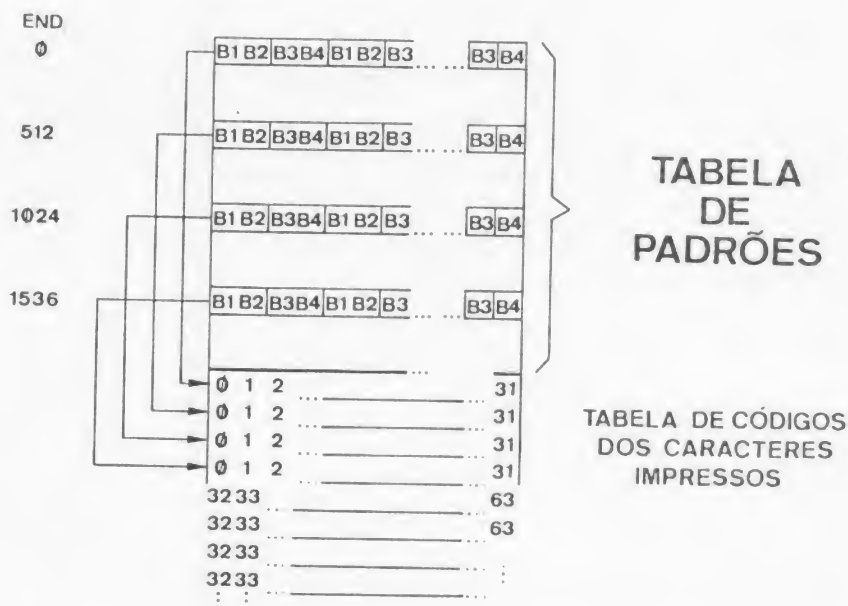
Tabela	Endereço	Linhas de tela
0	0-511	0- 4- 8-12-16-20
1	512-1023	1- 5- 9-13-17-21
2	1024-1535	2- 6-10-14-18-22
3	1536-2047	3- 7-11-15-19-23

No entanto, 1024 caracteres são mais do que suficientes para encher a tela sem que dois deles se repitam (768 bastam) e, então, apenas 192 caracteres são aproveitados de cada tabela (192x4=768).

A tabela de códigos dos caracteres impressos é única; mas para facilitar imaginemos que seja dividida em seis partes, cada uma responsável por quatro linhas consecutivas da tela (4x32=128 caracteres). As linhas 0 a 3 da tela são preenchidas com os caracteres de 0 a 31, um para cada coluna. As linhas 4 a 7 são preenchidas com os caracteres 32 a 63, e assim por diante.

Analise a figura 2.12:

FIGURA 2.12 - Organização da SCREEN 3.



TRANSFORMANDO A SCREEN 1 EM SCREEN 2

A quase totalidade dos jogos feitos para MSX usa como fundo a tela de 32 caracteres (SCREEN 1). O uso generalizado da SCREEN 1 deve-se à grande velocidade e facilidade de manuseio, além da baixa quantidade de memória exigida para definir seus caracteres.

Normalmente, a SCREEN 1 possui 256 caracteres definíveis com uma cor de frente e outra de fundo para cada oito caracteres ("octetos"). Com um pouco de técnica, a SCREEN 1 pode passar a usar 768 caracteres distintos e oito cores de frente e fundo para cada um de seus caracteres, aumentando a definição da imagem. Naturalmente, a memória usada na VRAM para este implemento é bem maior que a usada normalmente pela SCREEN 1.

Na verdade, essa "mágica" é facilmente explicável: ela consiste em fazer o computador "pensar" que está operando na SCREEN 1 enquanto o VDP opera no modo SCREEN 2. O resultado disto é podermos (literalmente) digitar, listar, imprimir e até "rolar" a tela inteira com as facilidades da SCREEN 1 e com a definição de imagens da SCREEN 2.

Basicamente, o que temos a fazer divide-se em dois passos. O primeiro é passar o computador para SCREEN 1, com isso todos os dados na VRAM e as variáveis de sistema ficam acertados para permitir a visualização dos caracteres, a digitação e a edição de programas. O segundo passo é mudar o VDP do modo SCREEN 1 para SCREEN 2 sem alterar as informações da memória. Isso poderia ser feito pelo comando VDP(N)=X, mas já existe para isto uma rotina do BIOS: SETGRP (&H7E).

Vamos então para um exemplo prático: Digite o programa da figura 2.13.

FIGURA 2.13 - Programa que transforma a SCREEN 1 em 2

```
10 COLOR 15,4,7
20 SCREEN 1
30 DEFUSR=&H7E:X=USR(0)
40 FOR I=0 TO 2:FOR J=0 TO 255:
50 VPOKE 6144+I*256+J,J:NEXT J,I
60 FOR I=0 TO 2047:A=PEEK(I+7103)
70 VPOKE I+2048,A
80 VPOKE I+4096,A
90 NEXT I:FOR I=8192 TO 14335
100 VPOKE I,31:NEXT I:END
```

Após executá-lo, veja como a tela 2 se parece com a tela 1. Digitando alguns comandos, você perceberá que o cursor só aparece no primeiro terço da tela (por que ?). Experimente agora alterar a tabela de cores (iniciada em 8192), com a seguinte alteração:

```
100 VPOKE I,255*RND(1):NEXT I:END
```

Depois de rodar o programa novamente (gostou do festival de cores ?), experimente outras alterações a seu gosto. Ao final do processo a VRAM passa a funcionar da seguinte maneira:

0-20476: tabela de formas para o primeiro terço da tela.
 2048-4095: tabela de formas para o segundo terço da tela.
 4096-6143: tabela de formas para o terceiro terço da tela.
 6144-6911: tabela dos códigos impressos na tela.
 8192-10239: tabela de cores para o primeiro terço.
 10240-12287: tabela de cores para o segundo terço.
 12288-14335: tabela de cores para o terceiro terço.

Além desses endereços, há também as tabelas dos sprites, que não sofrem alteração.

Vamos agora juntar os conhecimentos aprendidos até agora e transformá-los em algo mais útil.

Foi tratado no Capítulo 1 o funcionamento dos hooks, e se você der uma olhada no apêndice II verá que existem hooks de muitos comandos do BASIC, inclusive do comando SCREEN.

Então, por que não criarmos a SCREEN 4 ?

O programa SCREEN 4 (figura 2.14) quando executado, aponta o hook do comando SCREEN para o início da rotina. Assim sempre que o comando SCREEN for executado, o controle passará para o programa e serão feitas todas as inicializações para que o VDP gere a SCREEN 2, enquanto o Interpretador opera em SCREEN 1. Também o padrão do carácter 255 (cursor) é alterado na tabela de padrões do segundo e terceiro terço da tela para que o cursor seja visível por toda a tela. As três tabelas de cores são preenchidas com &HF0 (cor de fundo transparente e cor de frente branca).

O valor do topo do STACK será descartado, para que o controle volte ao Sistema Operacional, e não para a rotina do SCREEN, evitando assim, um erro de sintaxe. Afinal a sintaxe do comando SCREEN não aceita o parâmetro quatro normalmente.

FIGURA 2.14 - Programa SCREEN 4

	ORG	0D000H
006F	INIT32=EQU	006FH
007E	SETGRP=EQU	007EH
005C	LDIRVM=EQU	005CH
0072	INIRGP=EQU	0072H

```

0056      FILVRM= EQU 0056H
FFC0      HKSCRE= EQU 0FFC0H
63EA      END= EQU 63EAH
D000 3EC3      LD A,0C3H      ;APONTA HOOK DA
D002 32C0FF     LD (HKSCRE),A ;SCREEN P/ 0
D005 210CD0     LD HL,INIC    ;INICIO.
D008 22C1FF     LD (HKSCRE+1),HL
D00B C9        RET           ;RETORNA AO BASIC.
D00C
D00C FE15      INIC: CP 15H    ;VERIFICA SE # 3.
D00E C0        RET NZ        ;SE ( ) RETORNA PARA
                                ;ROTINA DE SCREEN
                                ;ARMAZENA REGISTRADORES.
D00F D9        EXX          ;INICIALIZA SCREEN 1.
D010 CD6F00     CALL INIT32   ;SETA VDP P/ SCREEN 2.
D013 CD7E00     CALL SETGRP   ;TRANSFERE TABELA
D016 21BF18     LD HL,18BFH   ;DE PADROES P/ 0
D019 110008     LD DE,800H    ;2º TERÇO DA
D01C 010008     LD BC,800H    ;TELA.
D01F CD5C00     CALL LDIRVM   ;TRANSFERE TABELA
D022 21BF18     LD HL,18BFH   ;DE PADROES P/ 0
D025 110010     LD DE,1000H   ;3º TERÇO DA
D028 010008     LD BC,800H    ;TELA.
D02B CD5C00     CALL LDIRVM   ;PREENCHE TODA A
D02E 210020     LD HL,2000H   ;TABELA DE CORES
D031 01FF17     LD BC,17FFH   ;COM BRANCO (FRENTE)
D034 3EF0       LD A,11110000B ;E INCOLOR (FUNDO).
D036 CD5600     CALL FILVRM   ;PREENCHE COM &HFF
D039 21F80F     LD HL,0FF8H   ;OS 8 BYTES QUE DEFI-
D03C 010800     LD BC,8       ;NEM O CURSOR NO
D03F 3EFF       LD A,0FFH     ;2º TERÇO.
D041 CD5600     CALL FILVRM   ;PREENCHE COM &HFF
D044 21F817     LD HL,17F8H   ;OS 8 BYTES QUE DEFI-
D047 010900     LD BC,9       ;NEM O CURSOR NO
D04A 3EFF       LD A,0FFH     ;3º TERÇO.
D04C CD5600     CALL FILVRM   ;RETORNA VALORES
D04F D9        EXX          ;DOS REGISTRADORES
D050 33        INC SP
D051 33        INC SP
D052 D7        RST 10H       ;PEGA PROX. CARACTERE
                                ;OU COMANDO DO BASIC
D053 C9        RET          ;RETORNA AO BASIC.
D054      END

```

Aplicando algumas das informações sobre os hooks (Capítulo 1) vamos elaborar um programa que, utilizando o hook do comando CMD do BASIC, (que tal qual o comando IPL não é implementado no sistema), altera o hook apontando-o para uma rotina de auxílio para a SCREEN 4.

FIGURA 2.15 - Programa CMD

```

005C      ORG 0C000H
C000 E5      LDIRVM= EQU 005CH
C001 D9      PUSH HL
C002 E1      EXX HL
C003 23      POP HL
C004 7E      INC HL
C005 E5      LD A,(HL)
C006 D630     SUB '0'
C008 210018     LD HL,B192-800H
C00B 110008     LD DE,800H
C00E 47      LD B,A
C00F      LOOP1:
C00F 19      ADD HL,DE
C010 10FD     DJNZ LOOP1
                                ;DA TELA.
                                ;ARMAZENA HL.
                                ;SALVA REGISTROS
                                ;RECUPERA HL.
                                ;INCREMENTA HL.
                                ;CARREGA EM A O CODIGO
                                ;ASCII DO Nº DA TABELA.
                                ;ARMAZENA HL.
                                ;OBTÉM O Nº DA TABELA.
                                ;END. DA TABELA DE CORES.
                                ;COMPRIMENTO DA TABELA.
                                ;LOOP QUE APONTA HL P/ A
                                ;TABELA DE CORES DE UM
                                ;DETERMINADO TERÇO
                                ;DA TELA.

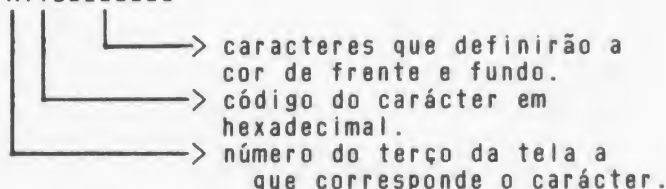
```

C012 EB	EX	DE,HL	;PAR 'DE' RECEBE END.
C013 E1	POP	HL	;DA TABELA DE CORES
C014 23	INC	HL	;RECUPERA HL E
C015 7E	LD	A,(HL)	;INCREMENTA.
C016 FE41	CP	'A'	;LE COD. DO 1º DIGITO.
C018 3802	JR	C,CONT	;TESTA SE E LETRA.
C01A D607	SUB	7	;SE AFIRMATIVO
C01C	CONT1:		;SUBTRAI 7.
C01C D630	SUB	'0'	;OBTEM VALOR CORRETO
C01E E60F	AND	00001111B	;OBTEM 1º NIBBLE.
C020 C827	SLA	A	;DESLOCA BITS
C022 C827	SLA	A	;P/ ESQUERDA.
C024 C827	SLA	A	
C026 C827	SLA	A	
C028 47	LD	B,A	;GUARDA 1º NIBBLE EM B.
C029 23	INC	HL	;INCREMENTA HL.
C02A 7E	LD	A,(HL)	;LE COD. DO 2º DIGITO.
C02B FE41	CP	'A'	;VERIFICA SE E LETRA.
C02D 3802	JR	C,CONT1	;SE AFIRMATIVO
C02F D607	SUB	7	;SUBTRAI 7.
C031	CONT1:		
C031 D630	SUB	'0'	;OBTEM VALOR DO
C033 80	OR	B	;2º NIBBLE.
C034 4F	LD	C,A	;JUNTA OS 2 NIBBLES.
C035 0600	LD	B,0	;TRANSFERE O VALOR
C037 E5	PUSH	HL	;PARA O PAR BC.
C038 C5	PUSH	BC	
C039 E1	POP	HL	
C03A 29	ADD	HL,HL	
C03B 29	ADD	HL,HL	
C03C 29	ADD	HL,HL	;MULTIPLICA POR 8.
C03D E5	PUSH	HL	
C03E C1	POP	BC	
C03F E1	POP	HL	
C040 7B	LD	A,E	;SOMA BC COM DE PARA
C041 81	ADD	A,C	;OBTER O ENDEREÇO ONDE
C042 5F	LD	E,A	;SE INICIA A TABELA
C043 3E00	LD	A,0	;DE CORES QUE INDICA AS
C045 8A	ADC	A,D	;CORES DE FRENTE E DE
C046 80	ADD	A,B	;FUNDO DE UM DETER-
C047 57	LD	D,A	;MINADO CARACTERE.
C048 23	INC	HL	;APONTA HL P/ A STRING.
C049 010B00	LD	BC,B	;CARREGA BC COM O
C04C CD5C00	CALL	LDIRVM	;TAMANHO DA STRING.
C04F 33	INC	SP	;COPIA STRING NA VRAM.
C050 33	INC	SP	;LIMPA STACK.
C051 D9	EXX		;RECUPERA OS REGISTROS.
C052 C5	PUSH	BC	;ARMAZENA BC.
C053 060C	LD	B,12	
C055	LOOP2:		
C055 23	INC	HL	;LOOP QUE APONTA HL
C056 10FD	DJNZ	LOOP2	;PARA O FINAL DO
C058 C1	POP	BC	;COMANDO.
C059 D7	RST	10H	;RECUPERA BC.
C05A C9	RET		;LE PROX. CARACTERE.
C05B	END		;RETORNA AO BASIC.

O programa CMD auxilia a SCREEN 4 na manipulação das tabelas de cores de um caracter.

Ele funciona da seguinte maneira. Quando o Interpretador chama o hook do comando CMD, o par HL aponta para o início dos parâmetros do comando. Você pode criar a sintaxe ou a função que desejar. No caso criamos a seguinte sintaxe para o CMD.

CMD"XYZzzzzzzzz"



Veja na figura 2.16 um exemplo de uso do CMD.

FIGURA 2.16 - Exemplo de uso do CMD

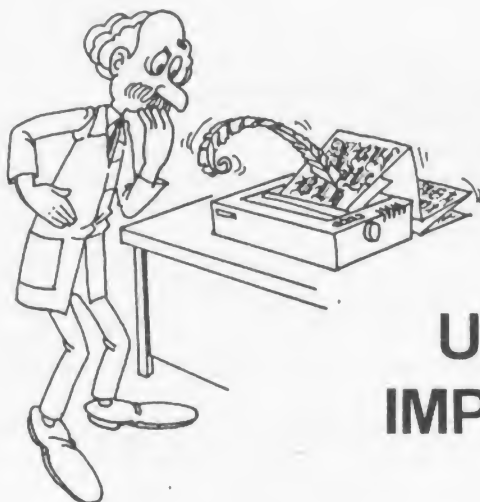
		cor de frente	cor de fundo	código do carácter	
1	█	12(&HC)	11(&HB)	&HCB	//
2	█	12(&HC)	11(&HB)	&HCB	//
3	█	12(&HC)	11(&HB)	&HCB	//
4	█	2	13(&HD)	&H2D	—
5	█	2	13(&HD)	&H2D	—
6	█	2	9	&H29)
7	█	15(&HB)	8	&HF8	°
8	█	15(&HB)	8	&HF8	°

O comando correspondente à figura 2.16 seria:

CMD"148////—)°°"

O valor que estiver no topo do STACK deve ser sempre descartado pois caso contrário, ao encontrar o primeiro comando de retorno (RET,RETZ,etc) a execução voltará para o comando CMD que nada mais faz que chamar a rotina de erro.

Fica como exercício fazer com que o comando CMD leia também variáveis tipo string.



USOS DA IMPRESSORA

Neste capítulo veremos como tirar melhor proveito da sua impressora e como compatibilizar o micro com o padrão de caracteres da impressora. Esse programa pode ser muito útil para os possuidores do Expert que utilizem uma impressora padrão ABICOMP.

Há também várias versões de cópia gráfica e um programa que transforma uma MÔNICA em uma impressora MSX.

COMPATIBILIZANDO O SEU MSX COM A IMPRESSORA

A maioria das impressoras nacionais possui os caracteres acentuados usados na língua portuguesa, respeitando o padrão ABICOMP. Porém se você reparar bem na tabela de caracteres da impressora e na tabela de caracteres do micro, verá que o código das letras acentuadas estão embaralhados, uns em relação aos outros.

Se você possui o Expert, coisas muito estranhas podem acontecer quando um carácter é enviado para a impressora. Já o HOTBIT faz as devidas conversões para o padrão ABICOMP.

Agora vamos ver o porquê dessas coisas estranhas acontecerem. O padrão ASCII (americano por sinal)

determina a função dos 127 primeiros caracteres do micro ou da impressora. Nas impressoras importadas, só haviam esses caracteres. Os de código maiores que 127 ou eram uma repetição dos 127 primeiros ou não apresentavam função específica. Isso porque a maioria dos micros só enviava 7 bits para a impressora, o que permite no máximo 128 códigos diferentes.

Com a explosão da microinformática no Brasil, foi criado o padrão ABICOMP, que nada mais é do que a redefinição dos caracteres maiores que 127 pelos caracteres especiais usados em nossa língua.

Mais recentemente (1986), foi criado o padrão ABNT (Associação Brasileira de Normas Técnicas), que é o padrão respeitado pelo Expert 1.1.

Os caracteres de &H80 a &H9F, porém, continuaram com a mesma função dos 32 primeiros (caracteres de controle) ou sem nenhuma função, dependendo do fabricante da impressora.

Em certos modelos de impressora, você pode, por exemplo, ativar o modo expandido enviando o carácter &H0E ou &H8E. Note que:

&H0E=&B000001101

&H8E=&B100001101

O carácter &H0E é o &H8E com o último bit setado e como naquela época a maioria dos micros só enviava 7 bits para a impressora, isso não tinha muita importância.

As acentuadas nos MSX começam no carácter &H7D e terminam no &HB9 e portanto, se compararmos as tabelas de caracteres do seu MSX e a de uma impressora que respeite o padrão ABICOMP ou ABNT, 32 caracteres (de &H7D a &H9F) não serão reconhecidos pela impressora como letras a serem impressas mas como comandos. A figura 3.1 apresenta a tabela de caracteres do Expert, do HOTBIT e de uma impressora padrão ABICOMP (no caso, uma MÔNICA PLUS).

Note que as três tabelas estão organizadas de maneiras diferentes: as dos MSX devem ser lidas linha por linha de cima para baixo, enquanto que a da MÔNICA deve ser consultada coluna por coluna da esquerda para direita.

O símbolo da "libra", por exemplo, na tabela do HOTBIT está na linha 9, coluna C. Portanto seu código em hexadecimal será &H9C. Na tabela da MÔNICA, este mesmo símbolo está na coluna B, linha C. Neste caso o seu código, em hexadecimal, será &HBC.

FIGURA 3.1A - Tabela de caracteres do Expert 1.0

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		☺	☻	♥	♦	♣	•	◼	◯	◐	♂	♀	♂	♂	♂	♂
1	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	Q	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	q	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	^
8	~	ü	é	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ä
9	é	æ	Æ	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö
A	á	í	ó	ú	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ
B	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä
C	—	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
D	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀
E	α	β	γ	π	Σ	σ	μ	τ	Φ	θ	Ω	δ	ω	φ	ε	π
F	≡	±	≥	≤	↑	↓	÷	×	•	•	•	•	•	•	•	•

FIGURA 3.1B - Tabela de caracteres do Expert 1.1 e HOTBIT 1.1

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		☺	☻	♥	♦	♣	•	◼	◯	◐	♂	♀	♂	♂	♂	♂
1	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	Q	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	q	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	^
8	Q	ü	é	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ä
9	é	æ	Æ	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö
A	á	í	ó	ú	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ
B	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä
C	—	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
D	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀
E	α	β	γ	π	Σ	σ	μ	τ	Φ	θ	Ω	δ	ω	φ	ε	π
F	≡	±	≥	≤	↑	↓	÷	×	•	•	•	•	•	•	•	•

*

EXPERT = {2

*

HOTBIT = {t

FIGURA 3.1C - Tabela da impressora MÔNICA da Elebra

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	SP	0	@	P	'	p			SP	ó	i	ò		
1	SOH	DC1	!	1	A	Q	a	q			A	ó	à	ó		
2	STX	DC2	"	2	B	R	b	r			A	o	á	ò		
3	ETX	DC3	#	3	C	S	c	s			A	õ	ä	ö		
4	EOT	DC4	\$	4	D	T	d	t			A	ö	ã	ô		
5	ENQ	NAK	%	5	E	U	e	u			A	æ	ä	æ		
6	ACK	SYN	&	6	F	V	f	v			C	U	c	ü		
7	BEL	ETB	'	7	G	W	g	w			E	U	e	ü		
8	BS	CAN	(8	H	X	h	x			E	O	é	ô		
9	HT	EM)	9	I	Y	i	y			E	U	e	ü		
A	LF	SUB	*	:	J	Z	j	z			E	Y	ë	ÿ		
B	VT	ESC	+	:	K	[k	[I		l	ß		
C	FF	FS	,	<	L	\	l	l			I	E	í	ä		
D	CR	GS	-	=	M]	m]			I	'	í	ö		
E	SO	RS	.	>	N	^	n	^			Y	\$	í	ü		
F	SI	US	/	?	O	_	o	_			N	o	ñ	±		

Existe também o fato de que nem sempre os caracteres acentuados do micro e da impressora representam a mesma letra acentuada. Por exemplo: os caracteres &H0E e &H8E ativam o modo expandido, e se você comandar

```
LPRINT "ó"
```

a impressora poderá começar a imprimir no modo expandido.

Uma maneira de se conseguir que algo seja impresso sem problemas, ou seja, com as devidas letras acentuadas, é colocar no lugar o caracter que possua o mesmo código da letra acentuada na impressora.

Por exemplo: No EXPERT 1.0 o "ç" possui o código 126, só que numa impressora padrão ABICOMP (no caso uma MÔNICA 6030) o "ç" possui o código 166. Portanto, para que a impressora possa imprimí-lo devemos enviar-lhe esse código, que corresponde ao carácter "ä". No texto, então, deveríamos escrever um "ä" no lugar de todo "ç".

Essa é uma maneira meio indecente de se fazer as coisas, mas felizmente funciona.

A outra maneira é interceptarmos o carácter que está sendo enviado para a impressora, através do hook da rotina LPTOUT do BIOS e verificar se ele é uma

letra acentuada, fazendo a devida conversão de acordo com o padrão da impressora que está sendo usada.

Vejamos o programa da figura 3.2 .

FIGURA 3.2 - Programa Filtro

C000 FE40	ORG 0C000H	
C002 2B13	CP 60H	;VERIFICA SE é "C".
C004 FE7E	JR Z,CHINU	;CASO AFIRM. SALTA.
	CP 7EH	;TESTA SE é MENOR
		;QUE &H7E.
C006 DB	RET C	;CASO AFIRM. RETORNA.
C007 FEB8	CP 0BBH	;VERIFICA SE é
		;CARACTERE GRAFICO.
C009 D0	RET NC	;CASO AFIRM. RETORNA.
C00A E5	PUSH HL	;ARMAZENA HL.
C00B D5	PUSH DE	;ARMAZENA DE.
C00C 219CBF	LD HL,TABCOV-7EH	;APONTA HL P/ TABELA.
C00F 1600	LD D,0	
C011 5F	LD E,A	;CODIGO DO CARACTERE.
C012 19	ADD HL,DE	;SOMA HL COM DE.
C013 7E	LD A,(HL)	;CARREGA A COM CODIGO
		;EQUIV. DA IMPRESSORA.
C014 D1	POP DE	;RESTAURA DE.
C015 E1	POP HL	;RESTAURA HL.
C016 C9	RET	;RETORNA AO BASIC.
C017	CHINU:	
C017 3EC6	LD A,0C6H	;CARREGA A COM "c".
C019 C9	RET	;RETORNA AO BASIC.
C01A	TABCOV:	
C01A A6207ED9	DEFB 0A6H,020H,07EH,0D9H,0CBH	
C01E C8		
C01F C3C4C1C5	DEFB 0C3H,0C4H,0C1H,0C5H,060H	
C023 60		
C024 C9CAC7CE	DEFB 0C9H,0CAH,0C7H,0CEH,0CDH	
C028 CD		
C029 CBA4A2A8	DEFB 0CBH,0A4H,0A2H,0ABH,0D5H	
C02D D5		
C02E B582B3B0	DEFB 0B5H,0B2H,0B3H,0B0H,0DBH	
C032 D8		
C033 D6DAD3D9	DEFB 0D6H,0DAH,0D3H,0D9H,020H	
C037 20		
C038 20202020	DEFB 020H,020H,020H,020H,0C2H	
C03C C2		
C03D CCD1D7CF	DEFB 0CCH,0D1H,0D7H,0CFH,0AFH	
C041 AF		
C042 DCDDE20	DEFB 0DCH,0DDH,0DEH,020H,020H	
C046 20		
C047 20202020	DEFB 020H,020H,020H,020H,020H	
C04B 20		
C04C A4C4AECE	DEFB 0A4H,0C4H,0AEH,0CEH,0B3H	
C050 B3		
C051 B3B9D920	DEFB 0B3H,0B9H,0D9H,020H,020H	
C055 20		
C056	END	

Note que esse programa pode compatibilizar o seu micro com qualquer padrão de impressora, desde que ela possua os caracteres acentuados.

Se sua impressora não respeitar o padrão ABIGCOMP, basta alterar a TABCOV do progama do modo descrito a seguir.

A tabela representa a sequência de caracteres acentuados do micro, a partir do caracter &H7E. Portanto no caso de um EXPERT 1.0, o primeiro elemento da tabela representaria o "C". Aí basta procurar qual o

código do "Ç" na sua impressora e colocá-lo como o primeiro elemento da tabela. E assim sucessivamente até o final dos caracteres acentuados.

TRANSFORMANDO UMA MÔNICA NUMA IMPRESSORA MSX

Alguns modelos da impressora MÔNICA da Elebra permitem a redefinição dos caracteres. Essa redefinição respeita alguns parâmetros que não são intuito deste livro descrever. Se houver interesse nos parâmetros dessa redefinição procure maiores informações no manual da MÔNICA.

A fonte de caracteres redefinidos ocupa uma parte do buffer da impressora e, para que a MÔNICA comece a imprimir com os caracteres redefinidos, é necessário ativar o modo Qualidade Carta com os caracteres &H1B (ESC) e "G" e o modo Fonte Alternativa com os caracteres &H1B (ESC) e "7". Para desativar, com os caracteres, &H1B, "H" (Qualidade Carta) e &H1B, "6" (Fonte Alternativa).

Se um caracter não for redefinido e a impressora estiver com a seleção de fonte alternativa ativa o caracter impresso terá o formato da fonte residente.

O programa a seguir gera um arquivo em disco, lendo a tabela de caracteres do seu MSX e definindo uma fonte para a MÔNICA, com os caracteres gráficos.

FIGURA 3.3 - Programa IMPMSX.BAS

```
10 *****
20 *   HENRIQUE DE FIGUEREDO LUZ   *
30 * IMPMSX-   JULHO -1986       *
40 *****
50 DEFINT A-Z:CLS
60 PRINT"1 .....Gerar arquivo fonte."
70 PRINT"2 .....Carregar fonte."
80 A$=INPUT$(1)
90 IF A$(<>"1" THEN GOTO 410
100 PRINT:PRINT"GERANDO ARQUIVO",,"AGUAR
DE..."
110 DATA 1,31,64,155,159,-33,168,175,-72
,184,192,-80,192,254,-31
120 OPEN"IMPMSX.ARQ" FOR OUTPUT AS#1
130 PRINT#1,27;91;
```

```

140 FOR Y=0 TO 4
150 READ IN,FI,CO
160 FOR T=IN TO FI
170 DIM D(7,7): P1=170:P2=170
180 FOR L=0 TO 7
190 IF PEEK(8*T+&H1BBF+L)=0 THEN 350
200 A$=BIN$(PEEK(8*T+&H1BBF+L))
210 IF LEN(A$)<>8 THEN A$="0"+A$:GOTO 210
220 Z=0
230 FOR C=7 TO 0 STEP -1
240 IF MID$(A$,C+1,1)="1" THEN D(L,C)=3+5*(C+1): IF Z=0 THEN D(L,C)=D(L,C)+128:Z=1
250 NEXT C
260 NEXT L
270 PRINT#1,CO+T;P1;P2;0;
280 FOR L=0 TO 7
290 FOR C=0 TO 7
300 IF D(L,C)<>0 THEN PRINT#1,D(L,C);
310 NEXT C:NEXT L
320 ERASE D:PRINT#1,0;:NEXT T
330 NEXT Y:PRINT#1,27;93
340 CLOSE#1:GOTO 400
350 IF L>3 THEN GOTO 380
360 P1=P1AND(255-(2^(7-L*2)))
370 GOTO 260
380 P2=P2AND(255-(2^(15-2*L)))
390 GOTO 260
400 PRINT"ARQUIVO GERADO.", "PRESSIONE QU
ALQUER TECLA":A$=INPUT$(1):RUN
410 POKE&HFFB6,201
420 BLOAD"IMPMSX.com"
430 CLS:PRINT"1 ....Expert versao 1.0"
440 PRINT"2 ....Expert versao 1.1"
450 PRINT"3 ....HOTBIT"
460 A$=INPUT$(1):IF A$<>"1" THEN GOSUB 640
470 POKE&HF417,255
480 OPEN"IMPMSX.ARQ" FOR INPUT AS#1
490 INPUT#1,L
500 LPRINT CHR$(L);
510 IF EOF(1) THEN CLOSE#1 ELSE GOTO 490
520 POKE&HFFB6,195
530 POKE&HFFB7,0

```

```

540 POKE&HFFB8,&HD0
550 IF A$="1" THEN POKE&HF417,0
560 LPRINT CHR$(27)"Q"CHR$(16)
570 LPRINT" @●♥♦◆•□◻▣▤▥▦▧▨▩*"
580 LPRINT"+!~|+|-_[]^X/\+";
590 FOR L=32TO 255
600 LPRINT CHR$(L);
610 NEXT L
620 LPRINT CHR$(27)"Q"CHR$(80)
630 PRINT"MICRO LIBERADO":END
640 DATA 7E,20,A6,D9,C8,C3,A2,C1,BB,C6,C
9,C0,B1,B7,A3,A9,B2,A1,A8,D5,B5,D2,D4,D0
,D8,D6,DA,B2,B9,20
650 DATA 20,20,20,20,C2
660 DATA CC,D1,D7,CF,AF,DC,DD,20,20,20,2
0,20,20,20,20,A4,C4,AE,CE,B3,D3,B9,D9
670 RESTORE 640: FOR L=0 TO 57:READ Z$:P
OKE &HD136+L,VAL("&H"+Z$):NEXT:L=POKE&HD12
1,&H60:RETURN

```

FIGURA 3.4 - Tabela de caracteres com fonte carregada.

Desse modo já conseguimos fazer a MÔNICA imprimir todos os caracteres gráficos do MSX. Agora vamos apresentar e comentar um programa que, tendo sido o hook da LPTOUT desviado para ele, gerencia a MÔNICA para ativar ou não a fonte de caracteres e imprimir o devido símbolo desejado.

FIGURA 3.5 - Programa IMPMSX.ASM

00A5		ORG	0D000H	
D000 3233D1	LPTOUT:	EQU	0A5H	
D003 D9		LD	(CARAC),A	; ARMAZENA CARACTERE
D004 3A2AD1		EXX		; ARMAZENA REGISTRADORES
D007 FEFF		LD	A,(FLAG)	; VERIFICA SE PRECEDIDO
D009 CA9AD0		CP	0FFH	;PELO CODIGO 1
D00C 3A2ED1		JP	Z,PRI32	;SALTA P/ PRI32
D00F FEFF		LD	A,(FLESC)	;VERIFICA SE PRECEDIDO
D011 2854		CP	0FFH	;POR ESC
		JR	Z,TESC	;CASO AFIRM. SALTA
				;P/ TESC
D013 3A2BD1		LD	A,(FLCOL)	;VERIFICA SE PRECEDIDO
D016 FEFF		CP	0FFH	;POR ESC "0"
D018 2872		JR	Z,RESCOL	;CASO AFIRM. SALTA
				;P/ RESCOL
D01A 3A33D1		LD	A,(CARAC)	; PEGA CODIGO CARACTERE
D01D FE01		CP	1	;VERIFICA SE VAI SER
				;ENVIADO CARACTERE
				;GRAFICO (QUE 32
D01F CACED0		JP	Z,SETFLG	;CASO AFIRM. SALTA
				;P/ SETFLG
D022 FE18		CP	18H	;VERIFICA SE É ESC
D024 CA93D0		JP	Z,SETESC	;CASO AFIRM SALTA
				;P/ SETESC
D027 FE20		CP	20H	;VERIFICA SE É CARACTERE
				;DE CONTROLE
D029 3837		JR	C,FIM	;CASO AFIRM SALTA P/ FIM
D02B FE7E		CP	07EH	;VERIFICA SE É CARACTERE.
				;GRAFICO) QUE &H7E
D02D D2D7D0		JP	NC,CGRA	;SE AFIRM. SALTA P/ CGRA
D030 FE60		CP	60H	;VERIFICA SE É "c"
D032 CA2D01		JP	Z,CHINU	;SE AFIRM. SALTA P/ CHINU
D035	TSTFFA:			
D035 21B6FF		LD	HL, 0FFB6H	;RESTITUI O HOOK
D038 36C9		LD	(HL),0C9H	;DA LPTOUT
D03A 2134D1		LD	HL, TAB2	;APONTA HL P/ ESC "H"
D03D 0604		LD	B,4	
D03F 3A2CD1		LD	A,(FL0C)	;VERIFICA SE LETRAS
D042 FEFF		CP	0FFH	;DEVEH SER IMPRESSAS
				;EH QUALIDADE CARTA
D044 2004		JR	NZ,CONTIN	;SE AFIRM. VAI P/ CONTIN
D046 23		INC	HL	;APONTA HL PARA
D047 23		INC	HL	;ESC '6'
D048 0602		LD	B,2	
D04A	CONTIN:			
D04A 3A2DD1		LD	A,(FLFA)	;VERIFICA SE F-A.
D04D FEFF		CP	0FFH	;ESTA ATIVADA
D04F 200C		JR	NZ,REST	;SE AFIRM. SALTA P/ REST
D051	LACO:			
D051 7E		LD	A,(HL)	;DESATIVA FONTE A.
D052 CDA500		CALL	LPTOUT	;E G.CARTA QUANDO
D055 23		INC	HL	;NECESSARIO
D056 10F9		DJNZ	LACO	
D058 212DD1		LD	HL,FLFA	;RESSETA FLAG DE
D05B 3600		LD	(HL),0	;FONTE A.
D05D	REST:			
D05D 21B6FF		LD	HL,0FFB6H	;APONTA HOOK DA LPTOUT
D060 36C3		LD	(HL),0C3H	;PARA ESTE PROGRAMA
D062	FIM:			
D062 D9		EXX		;RETOMA VALORES DOS
D063 3A33D1		LD	A,(CARAC)	;REGISTRADORES
D066 C9		RET		;VOLTA AO INTERPRETADOR
D067	TESC:			
D067 3A33D1		LD	A,(CARAC)	;VERIFICA SE FOI

D06A FE47	CP	'G'	ENVIADO ESC 'G'
D06C 2007	JR	NZ,CONT	SE AFIRM. SETA
D06E 212CD1	LD	HL,FL0C	FLAG DE QUALIDADE
D071 36FF	LD	(HL),0FFH	CARTA.
D073 1810	JR	TERESC	SALTA P/ TERESC
D075	CONT:		VERIFICA SE FOI
D075 FE48	CP	'H'	ENVIADO ESC 'H'
D077 2007	JR	NZ,CONTI	SE AFIRM. RESETA
D079 212CD1	LD	HL,FL0C	FLAG DE QUALIDADE
D07C 3600	LD	(HL),0	CARTA.
D07E 1805	JR	TERESC	SALTA P/ TERESC
D080	CONTI:		
D080 212BD1	LD	HL,FLCOL	PRESUME-SE QUE FOI
D083 36FF	LD	(HL),0FFH	ENVIADO ESC 'Q'
			FLCOL é SETADA P/QUE
			PROX. CARACTER SEJA EN-
			VIADO SEM INTERFERENCIA.
D085	TERESC:		
D085 212ED1	LD	HL,FLESC	RESETA FLAG
D088 3600	LD	(HL),0	DE ESC
D08A 18D6	JR	FIM	SALTA P/ FIM
D08C	RESCOL:		
D08C 212BD1	LD	HL,FLCOL	RESETA FLAG DE
D08F 3600	LD	(HL),0	COLUMNA
D091 18CF	JR	FIM	SALTA P/ FIM
D093	SETESC:		
D093 212ED1	LD	HL,FLESC	SETA FLAG
D096 36FF	LD	(HL),0FFH	DE ESC
D098 18C8	JR	FIM	SALTA P/ FIM
D09A	PRI32:		
D09A 212AD1	LD	HL,FLAG	RESETA FLAG DE CARACTERE
D09D 3600	LD	(HL),0	MEHOR QUE 32
D09F	IMPFA:		
D09F 212DD1	LD	HL,FLFA	VERIFICA SE
D0A2 7E	LD	A,(HL)	FONTE A. ESTA
D0A3 FEFF	CP	0FFH	ATIVA
D0A5 2888	JR	Z,FIM	SE AFIRM. SALTA P/ FIM
D0A7 36FF	LD	(HL),255	SETA FLAG DE FONTE A.
D0A9 21B6FF	LD	HL,0FFB6H	RESTITUI HOOK DA
D0AC 36C9	LD	(HL),0C9H	LPTOUT
D0AE 212FD1	LD	HL,INICTAB	APONTA HL P/ TABELA
D0B1 0605	LD	B,5	CARREGA B COM 5 P/ ENVIO.
D0B3 3A2CD1	LD	A,(FL0C)	SE IMPRESSORA JA
D0B6 FE00	CP	0	ESTIVER COM QUALIDADE
D0B8 2804	JR	Z,LOOP	CARTA ATIVADA NAO ENVIA
D0BA 23	INC	HL	ESC 'G' AVANCANDO
D0BB 23	INC	HL	2 ELEMENTOS DA TABELA.
D0BC 0603	LD	B,3	CARREGA B COM 3 P/ ENVIO.
D0BE	LOOP:		LOOP QUE ENVIA 3 OU
D0BE 7E	LD	A,(HL)	5 CARACTERES P/ QUE
D0BF CDA500	CALL	LPTOUT	SEJA IMPRESSO CARACTER
D0C2 23	INC	HL	GRAFICO MENOR QUE
D0C3 10F9	DJNZ	LOOP	32.
D0C5 21B6FF	LD	HL,0FFB6H	APONTA HOOK NOVAAMENTE
D0C8 36C3	LD	(HL),0C3H	PARA ESTE PROGRAMA.
D0CA 33	INC	SP	DESCARTA VALOR DO STACK.
D0CB 33	INC	SP	
D0CC 1894	JR	FIM	SALTA P/ FIM.
D0CE	SETFLG:		
D0CE 212AD1	LD	HL,FLAG	SETA FLAG QUANDO
D0D1 36FF	LD	(HL),0FFH	ENVIADO CARACTERE 1.
D0D3 33	INC	SP	DESCARTA VALOR DO STACK.
D0D4 33	INC	SP	
D0D5 1888	JR	FIM	SALTA P/ FIM
D0D7	CGRA:		
D0D7 3A33D1	LD	A,(CARAC)	CARREGA A COM CARACTERE
D0DA FEC0	CP	0C0H	VERIFICA SE > QUE &HC0.
D0DC 3018	JR	NC,CAMAC0	SE AFIRM. VAI P/ CAMAC0.
D0DE FE88	CP	088H	VERIFICA SE > QUE &H88.
D0E0 301E	JR	NC,CAIJGI	SE AFIRM. VAI P/ CAIJGI.
D0E2 FEB0	CP	080H	VERIFICA SE é
			LETRA ACENTUADA.
D0E4 3028	JR	NC,FILTRO	SALTA P/ FILTRO.
D0E6 FEAB	CP	0ABH	VERIFICA SE é > QUE &HAB.
D0E8 3010	JR	NC,INTERR	SE AFIRM SALTA P/ INTERR.

D0EA FEA0	CP	0A0H	VERIFICA SE é > QUE &HA0.
D0EC 3020	JR	NC, FILTRO	
D0EE FE9B	CP	9BH	VERIFICA SE é < QUE &H9B.
D0F0 381C	JR	C, FILTRO	SE AFIRM. SALTA P/FILTRO.
D0F2 D621	SUB	21H	CONVERTE O CARACTERE.
D0F4 3233D1	LD	(CARAC),A	ARMAZENA O CODIGO
			EM CARAC.
D0F7 18A6	JR	IMPFA	VAI IMPRIMIR EM F.A.
D0F9	CAMAC0:		
D0F9 D61F	SUB	1FH	CONVERTE CARACTERE.
D0FB 3233D1	LD	(CARAC),A	ARMAZENA O CODIGO
			EM CARAC.
D0FE 189F	JR	IMPFA	VAI IMPRIMIR EM F.A.
D100	CAIJGI:		
D100 D650	SUB	50H	CONVERTE CARACTERE.
D102 3233D1	LD	(CARAC),A	ARMAZENA O CODIGO
			EM CARAC.
D105 189B	JR	IMPFA	VAI IMPRIMIR EM F.A.
D107	INTERR:		
D107 D64B	SUB	4BH	CONVERTE CARACTERE.
D109 3233D1	LD	(CARAC),A	ARMAZENA O CODIGO
			EM CARAC.
D10C 1891	JR	IMPFA	VAI IMPRIMIR EM F.A.
D10E	FILTRO:		
D10E FE60	CP	60H	VERIFICA SE é 'c'.
D110 2810	JR	Z, CHINU	SE AFIRM. VAI P/CHINU.
D112 2133D1	LD	HL,TABCOV	APONTA HL P/ INICIO
			DA TABELA DE CONVERSAO.
D115 D67E	SUB	7EH	SUBTRAI &H7D.
D117 1600	LD	D,0	TRANSFERE VALOR
D119 5F	LD	E,A	DE A P/ DE.
D11A 19	ADD	HL,DE	SOMA COM HL.
D11B 7E	LD	A,(HL)	CARREGA A COM CARACTERE
			ACENTUADO DA IMPRESSORA.
D11C 3233D1	LD	(CARAC),A	ARMAZENA EM CARAC.
D11F C335D0	JP	TSTFFA	SALTA P/ TSTFFA.
D122	CHINU:		
D122 3EC6	LD	A,0C6H	CONVERTE NO 'c'
			DA IMPRESSORA.
D124 3233D1	LD	(CARAC),A	ARMAZENA EM CARAC.
D127 C335D0	JP	TSTFFA	SALTA P/ TSTFFA.
D12A 00	FLAG:	DEFB 0	
D12B 00	FLCOL:	DEFB 0	
D12C 00	FLQC:	DEFB 0	
D12D 00	FLFA:	DEFB 0	
D12E 00	FLESC:	DEFB 0	
D12F 18471837	INICTAB	DEFB 27,71,27,55	
D133 00	CARAC:	DEFB 0	
D134 18481836	TAB2:	DEFB 27,72,27,54	
D138 A6807E20	TABCOV:	DEFB 0A6H,80H,7EH,20H	
D13C C83C4C1		DEFB 0C8H,0C3H,0C4H,0C1H	
D140 C560C9CA		DEFB 0C5H,60H,0C9H,0CAH	
D144 2020CDCB		DEFB 20H,20H,0CDH,0CBH	
D148 A400A8D5		DEFB 0A4H,00,0A8H,0D5H	
D14C 85828380		DEFB 0B5H,0B2H,0B3H,0B0H	
D150 D8D620D3		DEFB 0DBH,0D6H,20H,0D3H	
D154 D9202020		DEFB 0D9H,20H,20H,20H	
D158 2020C2CC		DEFB 20H,20H,0C2H,0CCH	
D15C D1D7CFAF		DEFB 0D1H,0D7H,0CFH,0AFH	
D160 DCD0DE20		DEFB 0DCH,0DDH,0DEH,20H	
D164 20202020		DEFB 20H,20H,20H,20H	
D168 2020A4C4		DEFB 20H,20H,0A4H,0C4H	
D16C 2020B383		DEFB 20H,20H,0B3H,0B3H	
D170 2020B383		DEFB 20H,20H,0B3H,0B3H	
D174		END	

A maneira de se imprimir no vídeo um caract-
ter menor que 32 é comandar

PRINT CHR\$(1)CHR\$(64+n)

onde n é o código do carácter. Quando um desses caract-
teres é enviado para a impressora, tudo se passa da
mesma forma, desde que a variável do sistema NTMSXP

indique uma impressora MSX conectada ao micro.

Quando, por exemplo é comandado

```
LPRINT " @ "
```

e a variável do sistema estiver selecionada para uma impressora MSX, são enviados os caracteres &H01 e &H41, ou seja, o mesmo que comandar

```
LPRINT CHR$(1)"A"
```

Nos MSX é "default" o valor zero para a variável de sistema NTMSXP (&HF417), indicando que a impressora conectada é uma impressora MSX.

No Expert versão 1.1 e no HOTBIT, devido aos filtros internos que foram adaptados ao sistema operacional para compatibilizá-los com os padrões ABNT e ABICOMP respectivamente, o funcionamento da NTMSXP é inverso. Quando a NTMSXP está em 0, os filtros estarão ativos, e se diferente de zero o sistema operacional "pensará" que a impressora conectada é MSX.

Existe uma flag no programa chamada FL32 (Flag para caracteres menores que 32) que é ativada sempre que o caracter &H1, que não possui função na MÔNICA, é enviado. O caracter seguinte já terá o seu código correto, pois os caracteres menores que 32 foram redefinidos a partir do &H42 (66 em decimal), bastando apenas verificar se a fonte está ativa ou não.

Alguns comandos para a impressora, necessitam mais de um byte, e normalmente esses caracteres são precedidos pelo ESC (&H1B). Por isso teremos uma flag chamada FLESC (Flag de ESC) que será setada quando um ESC for enviado para impressora. Isso será feito para que os caracteres de controle, não sejam alterados pelo programa, pois eles não serão impressos, mas sim analisados pelo Sistema Operacional da impressora.

Quando a FLESC está setada, é analisado se o carácter enviado seguidamente é um "G" ou um "H", para que o programa saiba se deve desligar ou não o modo de Qualidade Carta ao final da impressão de um bloco de caracteres gráficos.

Para indicar se o modo de Qualidade Carta deve ficar permanentemente ativo, existe a flag FLQC (Flag de Qualidade Carta) que quando setada, impede que esse modo seja desligado. Quando ESC e um "H" são enviados, a FLQC é resetada.

Quando o carácter após o ESC não é um "G" ou um "H", ele é enviado sem nenhuma alteração, e a flag FLCOL (Flag do números de COLunas da impressora) é setada para que o próximo caracter também seja enviado para a impressora sem nenhuma alteração. Isso porque um grande número de comandos para a impressora utili-

zam três caracteres. Como, por exemplo, o comando para mudar o número de colunas da impressora.

Existe também a flag FLFA (FLAG de Fonte Alternativa), que será setada sempre antes de um bloco de caracteres gráficos ser enviado. Isso para evitar que os caracteres de controle para ativar a Qualidade Carta e a fonte de caracteres sejam enviados antes de cada caracter gráfico. Tal procedimento faria com que a MÔNICA, após cada caractere, voltasse o carro de impressão para imprimir o resto do caractere, pois a impressão de um caracter em qualidade carta é feita com passagem do carro de impressão duas vezes pelo mesmo lugar.

Após compilar o programa, grave-o com o nome de IMPMSX.COM, pois assim o programa IMPMSX.BAS o carregará automaticamente.

CÓPIA GRÁFICA NA IMPRESSORA

Quem leu o "Aprofundando-se no MSX" desta mesma Editora, deve estar lembrado de que nele haviam programas (em BASIC) que tiravam cópias da tela gráfica do computador na impressora. Aqueles que o testaram devem ter notado que ele era um pouco lento, devido ao enorme número de cálculos que realizava.

Pela proposta daquele livro (dirigido a programadores BASIC) não havia razão para colocar um programa em LM para obter o mesmo efeito. Agora, entretanto, vamos fazê-lo, apresentando-o em três versões sucessivas.

A primeira versão é meramente uma tradução do programa original, ou seja, o texto em BASIC foi traduzido para LM literalmente, linha por linha. É de fato a versão mais simples, e mesmo assim sua eficiência já é notável em relação ao programa em BASIC, graças à alta velocidade da LM. Experimente comparar a figura 3.6 com a figura 3.7.

FIGURA 3.6 - Programa cópia gráfica em BASIC

```
50000 REM ----- copy -----
50010 DATA 3E,00,CD,A5,00,C9
50012 FOR I = 0 TO 5:READ A$:POKE 51000
  I+I,VAL("&H"+A$):NEXT I:DEFUSR0=51000!
50020 LPRINT CHR$(27);CHR$(65);CHR$(8)
50040 FOR C=0 TO 31
50060 LPRINT CHR$(27);CHR$(75);CHR$(192
);CHR$(0);
50080 FOR L=23 TO 0 STEP -1
```

```

50100 FOR X=7 TO 0 STEP -1
50120 U=VPEEK((C*8+256*L)+X)
50122 V=VPEEK((C*8+256*L)+X+8192)
50125 IF V MOD 16 =CT THEN U=255
50130 POKE 51001!,U
50140 Z=USR0(0)
50160 NEXT X
50180 NEXT L
50200 LPRINT CHR$(10);
50220 NEXT C
50240 LPRINT CHR$(27);CHR$(65);CHR$(13)
50260 LPRINT CHR$(27);CHR$(81);CHR$(39)
50280 RETURN

```

FIGURA 3.7 - Programa cópia gráfica em LH

```

;-----
; COPIADOR DA SCREEN 2 VER. 1
; 1986 MILTON MALDONADO JR.
;-----

      ORG      0C000H
00A5  LPTOUT: EQU 0A5H
004A  RDVRM: EQU 04AH
0004  PAPEL: EQU 4
000F  TINTA: EQU 15
C000  INICIO:
C000 0E00      LD      C,0          ;C=Nº DA COLUNA.
C002  LOOP1:
C002 C5        PUSH    BC          ;SALVA C.
C003 CD64C0    CALL    SETGRA
C006 2E17      LD      L,23        ;L=Nº DA LINHA.
C008  LOOP2:
C008 E5        PUSH    HL          ;SALVA L.
C009 2607      LD      H,7         ;H=Nº DO BYTE
                                   ;DO CARACTERE.
C00B  LOOP3:
C00B E5        PUSH    HL          ;SALVA H.
C00C 55        LD      D,L         ;MULT. Lx256.
C00D 79        LD      A,C
C00E 17        RLA
C00F 17        RLA
C010 17        RLA                ;MULT. Cx8.
C011 5F        LD      E,A
C012 7C        LD      A,H
C013 6F        LD      L,A
C014 2600      LD      H,0
C016 19        ADD     HL,DE       ;BUSCA TABELA
                                   ;DE PADROES
C017 CD4A00    CALL    RDVRM       ;LE A URAM
C01A 32BFC0    LD      (BYTE),A   ;E SALVA VALOR.
C01D 110020    LD      DE,B192
C020 19        ADD     HL,DE
C021 CD4A00    CALL    RDVRM       ;LE TAB. CORES.
C024 47        LD      B,A
C025 E60F      AND     15
C027 FE0F      CP      TINTA
                                   ;COMP. COR DE
                                   ;FUNDO DO PON-
                                   ;TO COM A COR
                                   ;DE FRENTE DA
                                   ;TELA.

```

C029 2005	JR	NZ,SALTA1	
C02B 3EFF	LD	A,255	;SE IGUAL,
C02D 328FC0	LD	(BYTE),A	;ACENDE TODOS
			;OS PONTOS.
C030	SALTA1:		
C030 78	LD	A,B	
C031 1F	RRA		
C032 1F	RRA		
C033 1F	RRA		
C034 1F	RRA		
C035 E40F	AND	15	
C037 FE04	CP	PAPEL	;COMP. COR DE
			;FRENTE DO PON-
			;TO COM A COR
			;DE FUNDO DA
			;TELA.
C039 2004	JR	NZ,SALTA2	
C03B AF	XOR	A	;SE IGUAL,
C03C 328FC0	LD	(BYTE),A	;APAGA TODOS
			;OS PONTOS.
C03F	SALTA2:		
C03F 3A8FC0	LD	A,(BYTE)	
C042 CDA500	CALL	LPTOUT	;ENVIA BYTE.
C045 3B30	JR	C,STTXT	;SE OCORRER
			;FALHA,ABORTA.
			;RESTAURA H.
C047 E1	POP	HL	
C048 3EFF	LD	A,255	
C04A 25	DEC	H	
C04B BC	CP	H	
C04C 208D	JR	NZ,LOOP3	;FECHA LOOP3.
C04E E1	POP	HL	;RESTAURA L.
C04F 2D	DEC	L	
C050 BD	CP	L	
C051 20B5	JR	NZ,LOOP2	;FECHA LOOP2.
C053 3E0A	LD	A,10	
C055 CDA500	CALL	LPTOUT	;SALTA UMA
			;LINHA.
C058 3B1D	JR	C,STTXT	;SE OCORRER
			;FALHA,ABORTA'
			;RESTAURA C.
C05A C1	POP	BC	
C05B 0C	INC	C	
C05C 3E20	LD	A,32	
C05E B9	CP	C	
C05F 20A1	JR	NZ,LOOP1	;FECHA LOOP1.
C061 CD77C0	CALL	STTXT	
C064	SETGRA:		
C064 E5	PUSH	HL	;ATIVA IMPRES-
C065 C5	PUSH	BC	;SORA PARA MODO
C066 2182C0	LD	HL,GRA	;GRAFICO.
C069 0607	LD	B,7	
C06B	STLOOP:		
C06B 7E	LD	A,(HL)	;LOOP DE ENVIO
C06C CDA500	CALL	LPTOUT	;DOS BYTES P/
C06F 380F	JR	C,FIM	;A IMPRESSORA.
C071 23	INC	HL	
C072 10F7	DJNZ	STLOOP	
C074 C1	POP	BC	
C075 E1	POP	HL	
C076 C9	RET		
C077	STTXT:		
C077 E5	PUSH	HL	;ATIVA IMPRES-
C078 C5	PUSH	BC	;SORA PARA MODO
C079 2189C0	LD	HL,TEXT0	;TEXT0.
C07C 0606	LD	B,6	
C07E 18EB	JR	STLOOP	
C080	FIM:		
C080 C1	POP	BC	;ENCERRA LOOPS.
C081 C9	RET		;LIMPA PILHA.
C082	GRA:		
C082 1B410B1B	DEFB	27,65,8,27,75,192,0	
C086 4BC000			
C089	TEXT0:		
C089 1B410D1B	DEFB	27,65,13,27,81,39	
C08D 5127			
C08F	BYTE:		
C08F	DEFS	1	

Infelizmente os dois programas tem o incômodo de imprimir todas as figuras giradas de 90 graus. Isso ocorre porque no MSX os bytes que armazenam a tela indicam segmentos horizontais de imagem, e na impressora estes segmentos são impressos verticalmente. Na versão 2 (fig 3.8) esse inconveniente é eliminado pela inversão da ordem dos loops e por um algoritmo que cria uma matriz de 8 bytes girados de forma a compatibilizá-los com o padrão da impressora.

FIGURA 3.8 - Programa Cópia gráfica em LM versão 2

```

;-----
; COPIADOR DA SCREEN 2 VER. 2
; 1986 MILTON MALDONADO JR.
;-----
                                ORG    0C000H
00A5      LPTOUT=EQU 0A5H
0059      LDIRMV=EQU 059H
F3E9      FORCLR=EQU 0F3E9H
F3EA      BAKCLR=EQU 0F3EAH
004A      RDVRM=EQU 04AH
C000 AF    XOR    A
C001 3214C1 LD    (LOOPS),A      ;CONTA TERCOS
                                ;DA TELA.
C004      TBLOOP:
C004 AF    XOR    A
C005 3213C1 LD    (LOOPL),A      ;CONTA LINHAS.
C008      EXLOOP:
C008 AF    XOR    A
C009 3212C1 LD    (LOOPC),A      ;CONTA COLUNAS.
C00C CDE1C0 CALL   SETGRA
C00F      INLOOP:
C00F 210018 LD    HL,6144      ;BUSCA ENDERE-
C012 3A12C1 LD    A,(LOOPC)      ;CO DA TABELA
C015 4F      LD    C,A          ;DE NOMES (MA-
C016 0600    LD    B,0          ;PA DOS CARAC-
C018 09      ADD   HL,BC        ;TERES).
C019 3A13C1 LD    A,(LOOPL)
C01C 87      ADD   A,A
C01D 87      ADD   A,A
C01E 87      ADD   A,A
C01F 87      ADD   A,A
C020 87      ADD   A,A
C021 4F      LD    C,A
C022 09      ADD   HL,BC
C023 3A14C1 LD    A,(LOOPS)
C026 84      ADD   A,H
C027 67      LD    H,A
C028 CD4A00 CALL   RDVRM        ;LE CARACTERE.
C02B 6F      LD    L,A          ;BUSCA TABELA
C02C 2600    LD    H,0          ;DE FORMACAO
C02E 29      ADD   HL,HL        ;DO CARACTERE.
C02F 29      ADD   HL,HL
C030 29      ADD   HL,HL
C031 3A14C1 LD    A,(LOOPS)
C034 87      ADD   A,A
C035 87      ADD   A,A
C036 87      ADD   A,A
C037 84      ADD   A,H
C038 67      LD    H,A
C039 2210C1 LD    (PTR),HL      ;GUARDA POSI-
                                ;CAO 1º BYTE.
C03C 1115C1 LD    DE,BUFFER
C03F 010800 LD    BC,8
C042 CD5900 CALL   LDIRMV      ;COPIA OS 8
                                ;BYTES DO CA-
                                ;RACTERE.
C045 CD70C0 CALL   BITSB      ;ROTACIONA A
                                ;MATRIZ.

```

C048 3A12C1	LD	A, (LOOPC)	
C048 3C	INC	A	
C04C 3212C1	LD	(LOOPC), A	
C04F FE20	CP	32	
C051 20BC	JR	NZ, INLOOP	;FECHA LOOP COLUNAS.
C053 CD02C1	CALL	PR0D0A	
C056 3A13C1	LD	A, (LOOPL)	
C059 3C	INC	A	
C05A 3213C1	LD	(LOOPL), A	
C05D FE08	CP	8	
C05F 20A7	JR	NZ, EXLOOP	;FECHA LOOP LINHAS.
C061 3A14C1	LD	A, (LOOPS)	
C064 3C	INC	A	
C065 3214C1	LD	(LOOPS), A	
C068 FE03	CP	3	
C06A 2098	JR	NZ, TLOOP	;FECHA LOOP TERÇOS.
C06C CDEEC0	CALL	STTX	
C06F C9	RET		
C070	BITS:		;PROCEDIMENTOS ;DE 1 CARACTERE.
C070 0608	LD	B, B	
C072	TLOOP:		;ENVIA OS 8 ;BYTES DO CA- ;RACTERE.
C072 C5	PUSH	BC	
C073 CD80C0	CALL	BITS	
C076 3A0CC1	LD	A, (BYTE)	
C079 CDA500	CALL	LPTOUT	
C07C C1	POP	BC	
C07D 10F3	DJNZ	TLOOP	
C07F C9	RET		
C080	BITS:		;INICIA ROTACAO.
C080 AF	XOR	A	
C081 320DC1	LD	(LOOPB), A	;CONTA LINHAS/COLUNAS.
C084	XL00P:		
C084 3A0CC1	LD	A, (BYTE)	
C087 CB27	SLA	A	
C089 320CC1	LD	(BYTE), A	
C08C 3A0DC1	LD	A, (LOOPB)	
C08F 2115C1	LD	HL, BUFFER	
C092 0600	LD	B, 0	
C094 4F	LD	C, A	
C095 09	ADD	HL, BC	
C096 220EC1	LD	(BPOS), HL	;APONTA BYTE QUE ;ESTA SENDO PRO- ;CESSADO.
C099 2A10C1	LD	HL, (PTR)	
C09C 09	ADD	HL, BC	
C09D 010020	LD	BC, 8192	
C0A0 09	ADD	HL, BC	
C0A1 CD4A00	CALL	RDVRM	;LE TABELA DE CORES.
C0A4 E60F	AND	15	
C0A6 47	LD	B, A	
C0A7 3AE9F3	LD	A, (FORCLR)	
C0AA B8	CP	B	;COMP. COR DE ;FUNDO DO PONTO ;COM A COR DE ;FRENTE DA TELA. ;SE IGUAL, ACENDE ;O BIT.
C0AB 281A	JR	Z, SETB	
C0AD CD4A00	CALL	RDVRM	
C0B0 CB3F	SRL	A	
C0B2 CB3F	SRL	A	
C0B4 CB3F	SRL	A	
C0B6 CB3F	SRL	A	
C0B8 47	LD	B, A	
C0B9 3AEAF3	LD	A, (BAKCLR)	
C0BC B8	CP	B	;COMPARA COR DE ;FRENTE DO PONTO ;COM A COR DE ;FUNDO DA TELA. ;SE IGUAL, APAGA ;O BIT.
C0BD 2810	JR	Z, PROX	
C0BF 2A0EC1	LD	HL, (BPOS)	
C0C2 7E	LD	A, (HL)	
C0C3 E680	AND	128	
C0C5 2808	JR	Z, PROX	
C0C7	SETB:		;ACENDE PONTO A

C0C7 3A0CC1	LD	A, (BYTE)	; IMPRIMIR.
C0CA CBC7	SET	0, A	
C0CC 320CC1	LD	(BYTE), A	
C0CF	PROX:		; ENCERRA LOOP
C0CF 2A0EC1	LD	HL, (BPOS)	; DE ROTACAO.
C0D2 7E	LD	A, (HL)	
C0D3 17	RLA		
C0D4 77	LD	(HL), A	
C0D5 3A0DC1	LD	A, (LOOPB)	
C0D8 3C	INC	A	
C0D9 320DC1	LD	(LOOPB), A	
C0DC FE0B	CP	B	
C0DE 20A4	JR	NZ, XLOOP	
C0E0 C9	RET		
C0E1	SETGRA:		; ATIVA IMPRES-
C0E1 21F5C0	LD	HL, GRA	; SORA P/ MODO
C0E4 0607	LD	B, 7	; GRAFICO.
C0E6	STLOOP:		
C0E6 7E	LD	A, (HL)	
C0E7 CDA500	CALL	LPTOUT	
C0EA 23	INC	HL	
C0EB 10F9	DJNZ	STLOOP	
C0ED C9	RET		
C0EE	STTXT:		; ATIVA IMPRES-
C0EE 21FCC0	LD	HL, TEXTO	; SORA P/ MODO
C0F1 0606	LD	B, 6	; TEXTO.
C0F3 1BF1	JR	STLOOP	
C0F5	GRA:		
C0F5 1B410B1B	DEFB	27, 65, 8, 27, 75, 0, 1	
C0F9 4B0001			
C0FC	TEXT0:		
C0FC 1B410D1B	DEFB	27, 65, 13, 27, 81, 39	
C100 5127			
C102	PROD0A:		; IMPRIME SALTO
C102 3E0D	LD	A, 13	; DE LINHA.
C104 CDA500	CALL	LPTOUT	
C107 3E0A	LD	A, 10	
C109 C3A500	JP	LPTOUT	
C10C	BYTE:	DEFS 1	
C10D	LOOPB:	DEFS 1	
C10E	BPOS:	DEFS 2	
C110	PTR:	DEFS 2	
C112	LOOPC:	DEFS 1	
C113	LOOPL:	DEFS 1	
C114	LOOPPS:	DEFS 1	
C115	BUFFER:	DEFS 8	

Finalmente temos a versão 3 que, além "des-
virar" a figura, imprime-a de uma forma ampliada na
qual cada ponto da imagem tem seu tamanho dobrado. Uma
curiosidade desta versão é o modo de se ampliar cada
ponto da imagem. A ampliação horizontal é muito sim-
ples, pois para isso basta imprimir duas vezes cada
byte, saindo assim a largura dobrada. Para dobrar a
altura é mais complicado: deve-se partir o byte em
dois grupos de 4 bits (dois nibbles) e expandir cada
grupo para o dobro do seu tamanho.

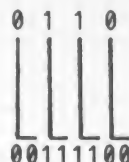
Por exemplo, imagine que o byte a ser enviado
para a impressora seja

&B01010110

Ao dividi-lo teremos dois nibbles que são
expandidos para dois bytes como ilustra a figura 3.9.

FIGURA 3.9 - Nibbles expandidos.

NIBBLES



NIBBLES EXPANDIDOS

00110011

00111100

É evidente que estes dois bytes não são enviados seguidamente para a impressora, bem como não são obtidos simultaneamente. Eles são processados em dois ciclos sucessivos que correspondem à metade superior e inferior de cada linha.

FIGURA 3.10 - Cópia gráfica em LM versão 3

```

;-----
; COPIADOR DA SCREEN 2 VER. 3
; 1986 MILTON MALDONADO JR.
;-----

      ORG    0C000H
00A5  LPTOUT: EQU 0A5H
0059  LDIRMV: EQU 059H
F3E9  FORCLR: EQU 0F3E9H
F3EA  BAKCLR: EQU 0F3EAH
004A  RDVRM: EQU 04AH
C000  INICIO:
C000  AF     XOR    A
C001  3263C1 LD     (TOGGLE),A      ;CONTA MEIAS
                                   ;LINHAS.
C004  AF     XOR    A
C005  326BC1 LD     (LOOPS),A      ;CONTA TERÇOS
                                   ;DA TELA.
C008  TBLOOP:
C008  AF     XOR    A
C009  326AC1 LD     (LOOPL),A      ;CONTA LINHAS.
C00C  EXLOOP:
C00C  AF     XOR    A
C00D  3269C1 LD     (LOOPC),A      ;CONTA COLUNAS.
C010  CD35C1 CALL    SETGRA
C013  INLOOP:
C013  21001B LD     HL,6144      ;BUSCA ENDERE-
C016  3A69C1 LD     A,(LOOPC)      ;CO DA TABELA
C019  4F     LD     C,A          ;DE NOMES (MA-
C01A  0600   LD     B,0          ;PA DOS CARAC-
C01C  09     ADD    HL,BC        ;TERES).
C01D  3A6AC1 LD     A,(LOOPL)
C020  87     ADD    A,A
C021  87     ADD    A,A
C022  87     ADD    A,A
C023  87     ADD    A,A
C024  87     ADD    A,A
C025  4F     LD     C,A
C026  09     ADD    HL,BC
C027  3A6BC1 LD     A,(LOOPS)
C02A  84     ADD    A,H
C02B  67     LD     H,A
C02C  CD4A00 CALL    RDVRM      ;LE CARACTERE.
C02F  6F     LD     L,A          ;BUSCA TABELA
C030  2600   LD     H,0          ;DE FORMACAO
C032  29     ADD    HL,HL        ;DO CARACTERE.
C033  29     ADD    HL,HL
C034  29     ADD    HL,HL
C035  3A6BC1 LD     A,(LOOPS)
C038  87     ADD    A,A
C039  87     ADD    A,A
C03A  87     ADD    A,A

```

C0C8 09	ADD HL,BC	
C0C9 2265C1	LD (BPOS),HL	;APONTA BYTE QUE
		;ESTA SENDO RODADO.
C0CC 2A67C1	LD HL,(PTR)	
C0CF 09	ADD HL,BC	
C0D0 010020	LD BC,8192	
C0D3 09	ADD HL,BC	
C0D4 CD4A00	CALL RDVRM	;LE TAB. CORES.
C0D7 E60F	AND 15	
C0D9 47	LD B,A	
C0DA 3AE9F3	LD A,(FORCLR)	
C0DD 88	CP B	;COMP. COR DE
		;FUNDO DO BYTE
		;COM A COR DE
		;FRENTE DA TELA.
		;SE IGUAL, ACENDE
		;0 BIT.
C0DE 281A	JR Z,SETB	
C0E0 CD4A00	CALL RDVRM	
C0E3 CB3F	SRL A	
C0E5 CB3F	SRL A	
C0E7 CB3F	SRL A	
C0E9 CB3F	SRL A	
C0EB 47	LD B,A	
C0EC 3AEAF3	LD A,(BAKCLR)	
C0EF 88	CP B	;COMP. COR DE
		;FUNDO DO BYTE
		;COM A COR DE
		;FRENTE DA TELA.
		;SE IGUAL, APAGA
		;0 BIT.
C0F0 2810	JR Z,PROX	
C0F2 2A65C1	LD HL,(BPOS)	
C0F5 7E	LD A,(HL)	
C0F6 E680	AND 128	
C0F8 2808	JR Z,PROX	
C0FA	SETB:	;ACENDE PONTO
		;A IMPRIMIR.
C0FA 3A60C1	LD A,(BYTE)	
C0FD CBC7	SET 0,A	
C0FF 3260C1	LD (BYTE),A	
C102	PROX:	
C102 2A65C1	LD HL,(BPOS)	;COLETA PROXIMO
C105 7E	LD A,(HL)	;BIT.
C106 17	RLA	
C107 77	LD (HL),A	
C108 3A64C1	LD A,(LOOPB)	
C10B 3C	INC A	
C10C 3264C1	LD (LOOPB),A	
C10F FE08	CP B	;ENCERRA LOOP.
C111 20A4	JR NZ,XLOOP	
C113 C9	RET	;RETORNA.
C114	AMP:	;AMPLIA UM BYTE.
C114 3262C1	LD (BYTE2),A	
C117 AF	XOR A	
C118 3261C1	LD (ABYTE),A	
C11B 0604	LD B,4	
C11D	LOOPA:	
C11D 3A62C1	LD A,(BYTE2)	;AVANCA UM BIT
C120 17	RLA	;EM BYTE2.
C121 3262C1	LD (BYTE2),A	
C124 3A61C1	LD A,(ABYTE)	;COLOCA O BIT
C127 17	RLA	;EM ABYTE, DUAS
C128 17	RLA	;VEZES.
C129 CB4F	BIT 1,A	
C12B 2802	JR Z,BIT0	
C12D CBC7	SET 0,A	
C12F	BIT0:	
C12F 3261C1	LD (ABYTE),A	
C132 10E9	DJNZ LOOPA	
C134 C9	RET	
C135	SETGRA:	;PASSA IMPRES-
C135 2149C1	LD HL,GRA	;SORA P/ MODO
C138 0607	LD B,7	;GRAFICO.
C13A	STLOOP:	
C13A 7E	LD A,(HL)	
C13B CDA500	CALL LPTOUT	
C13E 23	INC HL	
C13F 10F9	DJNZ STLOOP	
C141 C9	RET	

C142		STTXT:			;PASSA IMPRES-
C142	2150C1		LD	HL,TEXT0	;SORA P/ MODO
C145	0606		LD	B,6	;TEXT0.
C147	1BF1		JR	STLOOP	
C149		GRA:			
C149	1B410B1B		DEFB	27,65,8,27,75,0,2	
C14D	4B0002				
C150		TEXT0:			
C150	1B410D1B		DEFB	27,65,13,27,81,39	
C154	5127				
C156		PR0D0A:			;SALTA LINHA.
C156	3E0D		LD	A,13	
C158	CDA500		CALL	LPTOUT	
C158	3E0A		LD	A,10	
C15D	C3A500		JP	LPTOUT	
C160		BYTE:	DEFS	1	
C161		ABYTE:	DEFS	1	
C162		BYTE2:	DEFS	1	
C163		TOGGLE:	DEFS	1	
C164		LOOPB:	DEFS	1	
C165		BPOS:	DEFS	2	
C167		PTR:	DEFS	2	
C169		LOOPC:	DEFS	1	
C16A		LOOPL:	DEFS	1	
C168		LOOPS:	DEFS	1	
C16C		BUFFER:	DEFS	8	



OPERANDO O CASSETE EM ASSEMBLY

Neste capítulo discutiremos como o computador processa o armazenamento e a recuperação de dados da fita cassete, e veremos como utilizar as rotinas da ROM para efetuar leituras e gravações em linguagem de máquina.

Todas as operações do cassete efetuadas pelo MSX, utilizam as sub-rotinas do BIOS iniciadas entre as posições &H00E1 e &H00F5 da ROM. De fato, se analisarmos os comandos BLOAD, CLOAD, LOAD e INPUT *, veremos que todos eles utilizam as mesmas sub-rotinas, o mesmo ocorrendo com o BSAVE, CSAVE, SAVE e PRINT *.

CABEÇALHOS DE GRAVAÇÃO

Todas as gravações feitas pelo MSX são precedidas de um cabeçalho que identifica o tipo de informação enviada ao gravador e o nome especificado pelo usuário. Assim, quando o computador for ler novamente essas informações, ele saberá identificar o tipo dos dados e não haverá o perigo de carregar um bloco inválido (por exemplo, um programa em LM na área do BASIC).

O formato do cabeçalho é o mesmo para todos os tipos de dados e é exemplificado na figura 4.1.

FIGURA 4.1 - Formato do cabeçalho usado pelo MSX

XX XX XX XX XX XX XX XX XX XX B0 B1 B2 B3 B4 B5

Onde:

XX é o identificador do tipo de gravação:

BSAVE	XX=&HD0
CSAVE	XX=&HD3
SAVE	XX=&HEA
ARQUIVO	XX=&HEA

B0 a B5 guardam os caracteres do nome dado pelo usuário. Se o nome for mais curto, os bytes excedentes são preenchidos com espaços (&H20).

Por exemplo, quando digitamos

BSAVE "CAS:ROTINA",&HB000,&HC047,&HC000

o cabeçalho enviado à fita será como na figura 2.

FIGURA 4.2 - Cabeçalho usado pelo BSAVE.

D0	D0	D0	D0	D0	D0	D0	D0	D0	D0	D0	52	4F	54	49	4E	41	
											:	:	:	:	:	:	
											R	O	T	I	N	A	
tipo de gravação																	

Como se vê, os três parâmetros habitualmente dados no comando BSAVE não ficam no cabeçalho, como se poderia imaginar. Após o cabeçalho, então, vem o bloco de dados propriamente dito cujo formato depende do tipo da gravação:

BSAVE: os seis primeiros bytes do bloco de dados representam os três endereços que você digita após o nome. Do sétimo byte até o último, então, fica o programa (ou dados) em si, tal como se situa na RAM. Assim, o comando dado acima deve originar um bloco de dados como exemplificado pela figura 4.3.

FIGURA 4.3 - Início do bloco de dados do BSAVE.

00	80	47	C0	00	C0	41	42	10	40	1.
:	:	:	:	:	:	:	:	:	:	:	:	:
end.	end.	end.				programa ou dados						
inic.	final	exec.				(cópia da RAM)						

CSAVE: todo o bloco de dados é uma cópia fiel da RAM (como no BSAVE), evidentemente sem os seis bytes de

endereços. A cópia normalmente se inicia no endereço 32769 (&H8001) e termina na primeira sequência de três zeros consecutivos (fim do programa).

Tomemos como exemplo o programa da figura 4.4

FIGURA 4.4 - Exemplo de programa gravado pelo CSAVE

MEMÓRIA

```
10 PRINT
20 RUN
30 END
```

BYTES GRAVADOS

```
07 ] ep1
80 ]
0A ] n1
00 ]
91 token do PRINT
00 fl
0C ] ep1
80 ]
14 ] n1
00 ]
8A token do RUN
00 fl
13 ] ep1
80 ]
1E ] n1
00 ]
81 token do END
00 fl
00 ] fim de programa
00 ]
```

Quando se carrega um programa em BASIC pelo comando CLOAD, os dados da fita são colocados a partir dos bytes apontados pela variável de sistema TXTTAB (&HF676) até que se encontrem três bytes nulos, quando a leitura é encerrada. Antes de devolver o controle ao operador, entretanto, a sub-rotina CLOAD deve informar ao Interpretador o endereço do último byte lido para que este possa definir a área das variáveis. Se isso não fosse feito, haveria o perigo do computador definir as variáveis sobre o programa em BASIC, o que certamente seria desastroso.

Existem três variáveis do sistema que controlam as variáveis do BASIC:

```
VARTAB (&HF6C2)  
ARYTAB (&HF6C4)  
STREND (&HF6C6)
```

é nelas que o comando CLOAD escreve o último endereço lido. Com exceção da VARTAB, as demais (ARYTAB e STREND) têm seu valor mudado quando se cria uma nova variável. A VARTAB só muda quando se altera o tamanho do programa BASIC que está na memória.

SAVE e PRINT #: como os dois comandos funcionam da mesma maneira, eles são explicados juntamente nesta parte. Após o cabeçalho inicial, blocos de 256 bytes são enviados para a fita até todo o arquivo ter sido gravado. Evidentemente, é pouco provável que o último bloco contenha exatamente 256 bytes; o normal é que ele seja menor. Desse modo, os bytes que "sobraram" são preenchidos com o valor &H1A.

Antes de um bloco de dados ser enviado, é necessária a certeza de termos 256 bytes a salvar (ou que os bytes restantes sejam os últimos do programa ou arquivo).

Para isso, o computador monta um "buffer" de 256 bytes, que só é enviado à fita quando este está preenchido, ou quando se dá um comando CLOSE.

A diferença básica entre o SAVE "CAS:" e o CSAVE "nome" está na forma de envio dos dados. No CSAVE, os dados são codificados numa forma chamada "binário compactado", que na verdade é uma cópia da RAM. No SAVE "CAS:" as linhas de programa são enviadas ao cassete na forma ASCII, ou seja, letra por letra. Logicamente esse sistema é bem menos eficiente, pois salva um número muito maior de bytes para representar a mesma informação.

Um aspecto interessante é que se houver erro durante uma leitura, será mais fácil para o usuário "editar" uma linha que entrou defeituosa pelo LOAD do que pelo CLOAD, devido à redundância de informações inerente ao SAVE e LOAD. Por outro lado, o CSAVE utiliza um comprimento bem menor da fita, sendo então menos provável que um defeito físico da fita venha a interferir na leitura pelo CLOAD.

AS SUB-ROTINAS DO BIOS

Podemos agora falar das sub-rotinas do BIOS de tratamento do cassete. Elas ficam situadas na página 0 da ROM e estão indicadas a seguir pelo endereço de chamada (em Hexadecimal), nome e função.

&H00E1 TAPION: Aciona o motor do gravador, desativa a interrupção e aguarda o sinal de sincronismo ("header") da fita.

&H00E4 TAPIN: Lê um byte da fita e devolve seu valor no registro A (acumulador do Z80).

&H00E7 TAPIOF: Desliga o motor do gravador, reativa a interrupção e retorna.

&H00EA TAPOON: Aciona o motor do gravador, desativa a interrupção e envia o sinal de sincronismo ("header") para a fita. O valor do registro A determina a duração do sinal: se A valer 0, o sinal será curto; caso contrário, o sinal será longo.

&H00ED TAPOUT: Envia o byte contido no registro A do Z80 para o gravador cassete.

&H00F0 TAPOOF: Como no TAPIOF, desliga o motor, devolve a interrupção e retorna.

&H00F3 STMOTR: Controla o motor do gravador da seguinte maneira:

A=0: desliga

A=1: liga

A=255: inverte (liga-desliga,
desliga-liga)

Note como existe uma simetria perfeita entre as sub-rotinas de escrita e leitura. Veremos agora como se utilizam essas rotinas.

GRAVANDO UM BLOCO DE DADOS

Imagine que temos a mensagem "OS CYLONIOS INVADIRAO A TERRA!" e queremos enviá-la para a fita. Inicialmente, devemos acionar o gravador e enviar para a fita o sinal de sincronismo, que é essencial para a hora da leitura. Em seguida enviamos os dados propriamente ditos e finalmente encerramos a gravação. O programa a seguir faz exatamente isso.

Figura 4.5 - Gravação de uma mensagem em Assembly.

	ORG	0C00H	
00EA	TAPOON=	EQU	0EAH
00ED	TAPOUT=	EQU	0EDH
00F0	TAPOOF=	EQU	0F0H
C000 CDEA00	CALL	TAPOON	!ACIONA O GRAVADOR
			!E ENVIA O SINCRONISMO.
C003 2114C0	LD	HL, TEXTO	!CARREGA HL COM
			!O ENDEREÇO DA
			!MENSAGEM.


```

C006          LOOP:
C006 7E          LD    A, (HL)          ;CARREGA A COM UM
                                          ;CARACTERE DA
                                          ;MENSAGEM.
C007 A7          AND    A              ;TESTA FIM DA
                                          ;MENSAGEM.
C008 2808        JR     Z, FIM          ;SE FIM, SALTA.
C00A E5          PUSH  HL              ;SALVA HL.
C00B CDE000      CALL  TAPOUT          ;ENVIA O BYTE.
C00E E1          POP   HL              ;RESTAURA HL.
C00F 23          INC   HL              ;PROXIMA LETRA.
C010 1BF4        JR     LOOP           ;REINICIA LOOP.
C012          FIM:
C012 CDF000      CALL  TAP00F          ;ENCERRA A GRAVACAO
                                          ;E DESLIGA MOTOR.
C015 C9          RET                  ;RETORNA
C016          TEXTO:
C016 4F532043    DB    'OS CYLONIOS INVADIRAO '
C01A 594C4F4E
C01E 494F5320
C022 494E5641
C026 44495241
C02A 4F20
C02C 41205445    DB    'A TERRA !'
C030 52524120
C034 21
C035 00          DB    0

```

Digite o programa da figura 4.5, prepare o gravador para gravar e execute-o. O computador enviará um sinal curto para o gravador, tal qual um programa bem pequeno.

Agora que você tem a mensagem gravada na fita, deve estar pensando que existe um outro programa capaz de lê-la. E, de fato, existe mesmo, como é mostrado na figura 4.6 .

FIGURA 4.6 - Leitor de mensagens em Assembly

```

                                ORG    0C100H
00E1          TAPION=EQU    0E1H
00E4          TAPIN=EQU    0E4H
00E7          TAPIOF=EQU   0E7H
C100 CDE100    CALL  TAPION          ;LIGA MOTOR E
                                          ;AGUARDA SINCRONISMO.

C103          LOOP:
C103 CDE400    CALL  TAPIN          ;LE UM BYTE DA FITA.
C106 A7        AND    A              ;TESTA SE A=0.
C107 2804      JR     Z, FIM          ;SE FIM, SALTA.
C109 0398      OUT   (152), A        ;IMPRIME O CARACTERE.
C10B 1BF6      JR     LOOP           ;PROXIMO CARACTERE.
C10D          FIM:
C10D CDE700    CALL  TAPIOF          ;ENCERRA A LEI-
                                          ;TURA E DESLIGA
                                          ;O MOTOR.
C110 C9        RET                  ;RETORNA.

```

Uma curiosidade deste programa é o uso do comando OUT. Isso decorre do próprio hardware dos MSX, pelo qual o controle da VRAM é feito pelas portas 152 (&H98) e 153 (&H99). Você verá que um "OUT (152), A" equivale à chamada do comando CHPUT (&HA2), só que é muito mais veloz. Se você tentar trocar o OUT pela chamada do CHPUT, perceberá que o programa não funcio-

nará pois o CHIPUT é lento demais e tirará o micro do sincronismo com a fita.

Digite o programa da figura 4.6, volte a fita, coloque o gravador para ler a mensagem gerada pelo Leitor e aguarde alguns segundos, quando então deverá aparecer a mensagem na tela.

Os programas Enviador e Leitor são um exemplo didático de como usar as rotinas do BIOS para cassete. Veremos agora como usar o BIOS para coisas mais úteis.

OUTRAS APLICAÇÕES

Baseados no raciocínio dos programas acima, podemos, em princípio, ler e gravar o que bem entendermos na fita cassete. Uma aplicação interessante é a duplicação de programas gravados pelo comando BSAVE, que já foi explorada no livro "Aprofundando-se no MSX", mas não explicada em detalhes naquela publicação. De fato, o programa "Header" (capítulo 6) lá publicado é muito semelhante ao Leitor, com duas diferenças básicas: a primeira, é que o programa "Header" escreve o dado lido da fita numa área da RAM (e não na tela) e a segunda é que ele não espera um byte 0 para retornar (em vez disto, carrega 16 bytes quaisquer, que no caso compõem um cabeçalho), e depois mais seis bytes que contêm os endereços do BSAVE.

A parte em Assembly do programa Header está listado na figura 4.7 de uma forma mais clara do que a usada no "Aprofundando-se no MSX".

FIGURA 4.7 - Programa Header

00E1		ORG 40000	
00E4		TAPION: EQU 0E1H	
EA60 CDE100		TAPIN: EQU 0E4H	
		CALL TAPION	;AGUARDA PRI-
			;MEIRA PARTE.
EA63 D8		RET C	;RETORNA SE
			;HOUE ERRO.
EA64 2100C0		LD HL,0C000H	;1º ENDEREÇO.
EA67 0610		LD B,10H	;N=16 BYTES.
EA69	LOOP0:		
EA69 E5		PUSH HL	;SALVÁ HL E BC.
EA6A C5		PUSH BC	
EA6B CDE400		CALL TAPIN	;LE 1 BYTE.
EA6E C1		POP BC	;VOLTA HL E BC.
EA6F E1		POP HL	
EA70 D8		RET C	;RETORNA SE
			;HOUE ERRO.
EA71 77		LD (HL),A	;COLOCA NA
			;MEMORIA.
EA72 23		INC HL	;INC. ENDEREÇO.
EA73 10F4		DJNZ LOOP0	;PROX. BYTE.
EA75 CDE100		CALL TAPION	;AGUARDA SE-
			;GUNDA PARTE.
EA78 D8		RET C	;RETORNA SE
			;HOUE ERRO.
EA79 2110C0		LD HL,0C010H	;1º ENDEREÇO.
EA7C 0606		LD B,06H	

EA7E		LOOP1:		
EA7E E5		PUSH HL		;SALVA HL E BC.
EA7F C5		PUSH BC		
EA80 CDE400		CALL TAPIN		;LE 1 BYTE.
EA83 C1		POP BC		;VOLTA HL E BC.
EA84 E1		POP HL		
EA85 DB		RET C		;RETORNA SE ;HOUE ERRO.
EA86 77		LD (HL),A		
EA87 23		INC HL		;INC. ENDERECO.
EA88 10F4		DJNZ LOOP1		;PROX. BYTE.
EA8A C9		RET		;RETORNA.

Basicamente o programa lê dois blocos do cassette: um de 16 bytes, que é escrito a partir do endereço &HC000. Esse bloco é o cabeçalho propriamente dito, lido para que se obtenha o nome do programa. Do segundo bloco são os seis bytes de endereçamento do BSAVE e eles são armazenados nas posições seguintes.

Existe também um pequeno trecho em Basic que lê na memória esses bytes (função PEEK) e imprime na tela os valores convenientes (nome, tipo de arquivo e endereços, se existirem). Você seria capaz de escrever este trecho ?

Tecnicamente, entretanto, há um problema com esse trecho em Assembly: ele não desliga o gravador no fim da leitura (ou se houver uma parada). Assim, para corrigir isso, o trecho em Basic que segue a sua chamada desliga o motor com o comando MOTOR OFF, o que nada altera. Observe que os comentários foram omitidos na parte final da listagem anterior pois seriam uma repetição sumária dos comentários anteriores.

VERIFICAÇÃO DO BSAVE

Você já deve ter se lamentado por perder um programa gravado pelo BSAVE, simplesmente porque não tinha como verificar se ele estava bem gravado. Se o programa era de sua autoria, então, o seu gravador deve ter levado muitas pancadas nessas ocasiões.

Para evitar que você destrua seu gravador e consiga sempre ler seus programas gravados em BSAVE, apresentamos agora uma rotina que lê o programa da fita e o compara byte a byte com o que está na memória, indicando se há um erro ou não.

A princípio, a rotina procura os dez bytes que indicam o tipo do programa que está na fita. Ao encontrar o código do BSAVE (&HD0), o nome é lido e impresso na tela, e então termina o primeiro bloco. O segundo bloco começa a ser lido, iniciando pelos seis bytes de endereços.

Os quatro primeiros bytes marcam o início e o fim da verificação e os dois restantes são despreza-

dos. Em seguida, começa o confronto byte a byte.

A verificação prossegue até o último byte e, se houver alguma divergência, o endereço correspondente é impresso na tela. Em qualquer instante a verificação pode ser interrompida com o pressionamento das teclas CTRL+STOP.

FIGURA 4.10 - Verificador do BSAVE

```

                                ORG 0D000H
0098 VDPDAT=EQU 098H
0099 VDPADD=EQU 099H
006C INITXT=EQU 06CH
00A2 CHPUT=EQU 0A2H
00C6 POSIT=EQU 0C6H
00E1 TAPION=EQU 0E1H
00E4 TAPIN=EQU 0E4H
00E7 TAPIOF=EQU 0E7H
F857 BUF=EQU 0F857H
F85D FLE=EQU 0F85DH
D000 VERIFY:
D000 CD6C00 CALL INITXT ;ATIVA SCREEN 0.
D003 AF XOR A ;AJUSTA PRIMEI-
D004 D399 OUT (VDPADD),A ;RA POSICAO DA
D006 D399 OUT (VDPADD),A ;TELA.
D008 INICIO:
D008 CDE100 CALL TAPION ;AGUARDA SIN-
D008 3876 JR C,FIM1 ;CRONISMO.
;SE HA ERRO,
;SALTA.
D00D 060A LD B,10 ;LE OS 10 BYTES
D00F LOOPH: PUSH BC ;DE TIPO DE AR-
D00F C5 CALL TAPIN ;QUIVO.
D010 CDE400 POP BC
D013 C1 DJNZ LOOPH
D014 10F9 CP 0D0H ;SE TIPO DIFE-
D016 FED0 JR NZ,INICIO ;RENTE, TENTA
D018 20EE ;DE NOVO.
;LE E IMPRIME
;O NOME DO
;ARQUIVO.
D01A 0606 LD B,6
D01C LOOPN: PUSH BC
D01C C5 CALL TAPIN
D01D CDE400 JR C,FIM1
D020 3861 OUT (VDPDAT),A
D022 D398 POP BC
D024 C1 DJNZ LOOPN
D025 10F5 LD A,"'"
D027 3E3E ;IMPRIME '"')'.
D029 D398 OUT (VDPDAT),A
D02B D398 OUT (VDPDAT),A
D02D CDE100 CALL TAPION
;AGUARDA SIN-
;CRONISMO.
;LE OS 6 BYTES
;COM OS ENDE-
;REPOS DO BSAVE.
D030 2157F8 LD HL,BUF
D033 0606 LD B,6
D035 LOOP0: PUSH HL
D035 E5 PUSH BC
D036 C5 CALL TAPIN
D037 CDE400 POP BC
D03A C1 POP HL
D03B E1 LD (HL),A
D03C 77 ;COLOCA NA ME-
;MORIA.
;PROX. BYTE.
D03D 23 INC HL
D03E 10F5 DJNZ LOOP0
D040 3E2C LD A,"'"
D042 2A57F8 LD HL,(BUF)
D045 CDA9D0 CALL IMPHL
D048 D398 OUT (VDPDAT),A
D04A 2A59F8 LD HL,(BUF+2)
D04D CDA9D0 CALL IMPHL

```

D050 D398	OUT	(VDPDAT),A	
D052 2A58F8	LD	HL,(BUF+4)	
D053 CDA9D0	CALL	IMPHL	
D058 3E20	LD	A,32	;IMPRIME ESPA-
D05A 0612	LD	B,18	;COS.
D05C	LOOPS:		
D05C D398	OUT	(VDPDAT),A	
D05E 10FC	DJNZ	LOOPS	
D060 2A57F8	LD	HL,(BUF)	;INICIA COMPA-
D063 ED5859F8	LD	DE,(BUF+2)	;RACAO.
D067 13	INC	DE	
D068 AF	XOR	A	
D069 325DF8	LD	(FLE),A	
D06C	LOOP:		;LOOP DE COMPA-
D06C E5	PUSH	HL	;RACAO.
D06D D5	PUSH	DE	
D06E CDE400	CALL	TAPIN	;LE UM BYTE.
D071 D1	POP	DE	
D072 E1	POP	HL	
D073 380E	JR	C,FIM1	
D075 BE	CP	(HL)	;COMPARA COM A
			;MEMORIA.
D076 C4A2D0	CALL	NZ,IMPDAT	;SE HA ERRO,
			;IMPRIME EN-
			;DERECO E O
			;DADO.
			;PROXIMO BYTE.
D079 23	INC	HL	
D07A A7	AND	A	
D07B E5	PUSH	HL	
D07C ED52	SBC	HL,DE	;TESTA SE TER-
D07E E1	POP	HL	;MINOU.
D07F 20EB	JR	NZ,LOOP	
D081 1805	JR	FIM	;FIM DA COMPA-
			;RACAO.
D083	FIM1:		
D083 3E01	LD	A,1	;ACENDE FLAG
D085 325DF8	LD	(FLE),A	;DE ERROS.
D088	FIM:		
D088 211000	LD	HL,16	;POSICIONA CUR-
D08B CDC600	CALL	POSIT	;SOR.
D08E CDE700	CALL	TAPIOF	;ENCERRA LEITU-
			;RA DA FITA.
			;TESTA FLAG DE
			;ERRO.
D091 3A5DF8	LD	A,(FLE)	
D094 A7	AND	A	
D095 C0	RET	NZ	
D096 21FCD0	LD	HL,MENS	;SE NAO HA ERRO,
D099	LOOPH:		;IMPRIME MENSA-
D099 7E	LD	A,(HL)	;GEM.
D09A A7	AND	A	
D09B C8	RET	Z	
D09C 23	INC	HL	
D09D CDA200	CALL	CHPUT	
D0A0 18F7	JR	LOOPH	
D0A2	IMPDAT:		
D0A2 CDA9D0	CALL	IMPHL	;ROTINA DE IM-
D0A5 CDB4D0	CALL	IMP	;PRESSAO DO
D0A8 C9	RET		;BYTE DIVERGENTE.
D0A9	IMPHL:		
D0A9 F5	PUSH	AF	;IMPRIME O ENDE-
D0AA 7C	LD	A,H	;RECO (VALOR HL).
D0AB CDC7D0	CALL	IMPA	
D0AE 7D	LD	A,L	
D0AF CDC7D0	CALL	IMPA	
D0B2 F1	POP	AF	
D0B3 C9	RET		
D0B4	IMPDI:		
D0B4 F5	PUSH	AF	
D0B5 3E2D	LD	A,'-'	
D0B7 D398	OUT	(VDPDAT),A	
D0B9 F1	POP	AF	
D0BA CDC7D0	CALL	IMPA	
D0BD 3E20	LD	A,32	
D0BF D398	OUT	(VDPDAT),A	
D0C1 3E01	LD	A,1	;ACENDE FLAG
D0C3 325DF8	LD	(FLE),A	;DE ERRO.

```

D0C6 C9          RET
D0C7             IMPA:      ;ROTINA QUE
D0C7 F5          PUSH AF      ;IMPRIME O
D0C8 C83F        SRL A        ;VALOR DO
D0CA C83F        SRL A        ;ACUMULADOR.
D0CC C83F        SRL A
D0CE C83F        SRL A
D0D0 01ECD0      LD BC,BASE
D0D3 81          ADD A,C
D0D4 4F          LD C,A
D0D5 3E00        LD A,0
D0D7 88          ADC A,B
D0D8 47          LD B,A
D0D9 0A          LD A,(BC)
D0DA D398        OUT (VDPDAT),A ;19. DIGITO.
D0DC F1          POP AF
D0DD E60F        AND 15
D0DF 01ECD0      LD BC,BASE
D0E2 81          ADD A,C
D0E3 4F          LD C,A
D0E4 3E00        LD A,0
D0E6 88          ADC A,B
D0E7 47          LD B,A
D0E8 0A          LD A,(BC)
D0E9 D398        OUT (VDPDAT),A ;29. DIGITO.
D0EB C9          RET
D0EC             BASE:
D0EC 30313233    DEFB '0123456789ABCDEF'
D0F0 34353637
D0F4 38394142
D0F8 43444546
D0FC             MENS:
D0FC 0D0A5345    DEFB 13,10,'SEM ERROS',13,10,0
D100 4D204552
D104 524F530D
D108 0A00

```

Para executar o verificador, você deverá utilizar os comandos DEFUSR=&HC000 e PRINT USR(0) ou então tente implementá-lo como comando alterando o hook do comando IPL (que não é usado pelo sistema).

AS VELOCIDADES DE GRAVAÇÃO

Uma curiosidade do sistema de cassete do MSX é a possibilidade de se alterar a velocidade da gravação por software. De fato, existem duas variáveis do sistema que indicam a velocidade com que um bit é enviado à fita:

LOW (&HF406-&HF407):

Estes dois bytes determinam a duração do pulso correspondente a um bit 0. O primeiro byte indica a duração da parte baixa do pulso, e o segundo dá a parte alta do mesmo.

HIGH (&HF408-&HF409):

O mesmo que LOW, só que para o bit 1.

Quando o computador é inicializado, a velocidade de gravação é ajustada para 1200 BPS (bits por

segundo), e os valores iniciais dessas variáveis são:

LOW:	&HF406	83 (decimal)
	&HF407	92 (decimal)
HIGH:	&HF408	38 (decimal)
	&HF409	45 (decimal)

E os valores para 2400 BPS são:

LOW:	&HF406	37 (decimal)
	&HF407	45 (decimal)
HIGH:	&HF408	14 (decimal)
	&HF409	22 (decimal)

Existe também uma variável chamada HEADER (&HF40A), que indica o comprimento (em bits) do sinal agudo ("header") que antecede as gravações. Normalmente, seu valor é de 15 (para 1200 BPS) e de 31 (para 2400 BPS). O número de bits enviados é igual ao conteúdo de HEADER multiplicado por 128. Assim, o "header" possui 1920 bits de comprimento para 1200 BPS, e 3968 bits para 2400 (evidentemente, aproximações de 2000 bits e 4000 bits respectivamente). Note que esta diferença de comprimento é devida à diferença de duração de um bit individualmente, que no segundo caso é metade do primeiro, e com isto os dois "headers" têm a mesma duração global. Trocando em miúdos, o "header" pode ser entendido como um gigantesco byte em que todos os bits estão "setados" (em nível 1).

O fato mais marcante do sistema cassete MSX é o modo como ele lê as gravações: as leituras são assíncronas, ou seja, quem determina a velocidade de leitura não é o computador, mas a própria gravação. De fato, se você gravar um programa com as variáveis LOW e HIGH alteradas (e portanto com velocidade alterada), conseguirá ler esses dados sem problemas, mesmo que as duas variáveis tenham sido restauradas (ou até alteradas para valores absurdos). Isto é muito vantajoso em termos de confiabilidade, já que, contrariamente a outros sistemas, as variações de rotação de um gravador não afetam a leitura dos dados.

Após conhecer este sistema tão flexível, você já deve estar pensando em alterar as variáveis LOW e HIGH para um valor muito menor com o objetivo de aumentar a velocidade de gravação. Teoricamente isso é possível até mais de 5000 BPS, mas é importante atentar para dois fatos:

1. Não é qualquer gravador que consegue gravar sinais tão rápidos, e qualquer fita retê-los. Um Data-Corder da Gradiente foi testado com fita cromo e superou a marca dos 4000 BPS com uma grande gama de volumes e 100 por cento de acerto, o que é muito bom. Mas isto não é tudo.

2. O filtro de áudio dos MSX é dimensionado para bloquear sinais mais lentos que 1200 BPS e mais velozes que 2400 BPS. Trata-se de um filtro passa-banda que serve para reduzir as interferências, mas que também limita a velocidade de gravação. A resposta desse filtro é implacável, atestando a boa qualidade conferida por seus fabricantes.

Percebe-se que aumentar a velocidade de gravação não é tão simples, pois torna-se necessário possuir conhecimentos de Eletrônica e "meter a mão na massa" alterando o filtro passa-banda do micro. Se você tentar (e conseguir) fazê-lo, escreva-nos contando sua experiência.





O SISTEMA DE CARTUCHO

Neste capítulo veremos basicamente, como o sistema MSX inicializa e reconhece o que está conectado em cada um dos seus slots; como trabalha com os cartuchos de programas; onde estão localizadas as variáveis que o MSX utiliza para armazenar as informações sobre cada slot; como funcionam os inicializadores de jogos para jogos que estavam em cartucho e como relocar programas em Assembly para as páginas 0 e 1 de um slot no qual existe memória RAM e como implementar o comando CALL através do BASIC.

Para o leitor menos familiarizado com o sistema MSX, recomendamos que antes de ler este capítulo, leia ou revise os conceitos básicos sobre slots e páginas de memória no livro "Aprofundando-se no MSX", desta mesma Editora.

INICIALIZAÇÃO DO SISTEMA

Toda vez em que você liga seu computador MSX, ele começa procurando "onde" está contida a memória RAM em seu sistema, sendo que essa procura é feita tanto nos slots primários, quanto nos secundários.

Essa procura de memória RAM é inicializada testando-se, em primeiro, lugar a região de memória &HBFFF; decrescendo até o endereço &H8000 e, após ter

achado um slot que a contenha, habilita essa página para poder usá-la. Continua à procura de memória RAM, repetindo o processo descrito acima, só que desta vez, fazendo teste na memória que vai do endereço &HFFFF decrescendo até o endereço &HC000 e também a habilita para ser usada.

Entretanto, mesmo que o sistema já tenha localizado em qual slot, e em que página existe memória RAM, o MSX irá fazer outra busca de memória RAM, só que agora, procurando um bloco contínuo de memória que possa ser habilitado desde o endereço &HFFFF até &H8000. Em seguida, será feito um teste para habilitar definitivamente um bloco de memória que estiver "mais perto" do slot 0 ou, no próprio slot 0, mas desde que esse esteja expandido.

Após esses breves mas complicados testes, o MSX irá fazer todo o resto da sua inicialização, como por exemplo, a configuração das variáveis do sistema para, logo em seguida, começar a procurar os cartuchos de programas e passar a executá-los.

RECONHECIMENTO DE CARTUCHOS

Um cartucho conectado em um dos slots do MSX pode ser reconhecido, basicamente, através dos dois primeiros bytes da memória desse mesmo cartucho, através de uma pequena procura feita pelo sistema após ser ligado e ter feito os testes que foram descritos acima.

Na realidade, existem vários bytes que ajudam o MSX a reconhecer um cartucho de programas.

A figura 5.1 apresenta, esquematicamente, esses bytes.

IDENTIFICATION - IDENTIFICAÇÃO

São usados nestes dois primeiros bytes os valores &H41 e &H42 ("A" e "B") para identificar um cartucho de programas ROM de páginas vazias.

INIT - INÍCIO

Estes bytes contém o endereço para o procedimento de inicialização para o cartucho que vai ser executado. No caso de NAO se desejar uma inicialização desse mesmo cartucho, deve ser colocado nestes bytes, o valor 00. Tais bytes só são usados caso NAO se deseje trabalhar cooperativamente com o Interpretador BASIC, como por exemplo, nos jogos em Assembly.

terpretador BASIC. Caso não se queira, deve ser colocado nestes bytes, o valor 00.

Quando o Interpretador BASIC do MSX encontra o comando CALL <NOME>, ele irá colocar o nome com que foi chamado o dispositivo na área de sistema para, logo em seguida, executar o endereço contido nestes bytes. Para usar esse tipo de chamada através do BASIC, temos um procedimento a ser seguido, que é explicado abaixo :

1 - O par de registradores HL realiza função de apontador de texto.

2 - O cartucho tem que estar habilitado da posição de memória que vai de &H4000 até &H7FFFH.

3 - A sintaxe para se poder usá-lo é:

CALL <NOME> [<ARGUMENTO>]

4 - O nome é estocado na área de sistema, terminado com 00. Deve-se lembrar que o buffer para o nome tem um comprimento fixo e máximo de 16 bytes, mas o nome não pode ter mais do que 15 caracteres.

5 - Se não existir o nome no cartucho com que foi chamado o programa, o mesmo deverá retornar ao BASIC com a CARRY FLAG setada e o apontador de texto (HL) descarregado.

6 - Se existir o nome no cartucho (ou no programa), este executará o programa, atualizará o apontador de texto (HL) para o fim do nome (usualmente, apontando para o byte 00 que indica fim de linha, ou "!" que indica fim de nome (declaração), e retorna com a CARRY FLAG resetada. No caso de se usar os argumentos, o apontador de texto é carregado com o endereço que aponta para o primeiro caracter não branco (00) após o nome.

NOTA : TODOS OS REGISTRADORES PODEM SER DESTRUÍDOS, EXCETO O REGISTRADOR [SP].

DEVICE - EXPANSÃO DE DISPOSITIVOS

Estes bytes contém o endereço de execução do programa, o qual contém uma rotina para a manipulação de um dispositivo, se este estiver contido no cartucho. Caso não exista um dispositivo (e um programa pa-

ra esta manipulação), estes bytes deverão conter o valor 00.

Caso exista um dispositivo, o BASIC irá executar esse endereço de acordo com o nome que estiver na área de sistema. Para usar mais essa facilidade que o BASIC MSX nos permite, temos o seguinte procedimento:

1 - O cartucho tem que estar habilitado, da posição de memória que vai de &H4000 até &H7FFF.

2 - O nome do dispositivo é estocado na área de sistema, terminado com 00. O buffer para esse nome tem comprimento fixo de 16 bytes, mas o nome do dispositivo não pode ser maior do que 15 caracteres.

3 - Um cartucho de 16 Kbytes pode ter até 4 dispositivos lógicos.

4 - Quando o BASIC encontra o nome de um dispositivo que ele mesmo não reconhece, ele chama o sistema de expansão de dispositivos (SOFTWARE) com o valor FFH no registrador A. Se o manuseio para aquele dispositivo não estiver implementado no cartucho, ele irá retornar com a CARRY FLAG setada. Entretanto, se estiver implementado, a identificação (ID) do dispositivo (de 0 a 3) será retornada no registrador A, e a CARRY FLAG resetada.

5 - Uma operação de arquivamento é realizada quando o cartucho reconhece um dispositivo próprio, e essa operação é especificada pelos seguintes valores passados pelo registrador A :

- 0 - Open
- 2 - Close
- 4 - I/O randômico
- 6 - Saída sequencial
- 8 - Entrada sequencial
- 10 - Função LOC
- 12 - Função LOF
- 14 - Função EOF
- 16 - Função FPOS
- 18 - Copia um caracter

A identificação do dispositivo é passada na variável de sistema "DEVICE".

TEXT - TEXTO

Estes bytes contém o endereço do texto BASIC (tokenizado) contido no cartucho. Caso você não queira colocar um programa em BASIC no cartucho, estes bytes devem conter o valor 00.

Caso exista o programa BASIC, o sistema irá pegar estes dois bytes como endereço inicial do texto BASIC, setará o apontador e iniciará a sua execução.

Para implementar o texto BASIC em cartucho, sugerimos que sejam seguidas as regras abaixo:

1 - Quando existir mais do que um cartucho semelhante conectado em um dos slots, o sistema irá executar o cartucho de programas que estiver "mais próximo" do slot 0.

2 - O cartucho deve estar habilitado para que seja executado entre os endereços &H8000 até &HBFFF e o programa BASIC não poderá exceder a 16 Kbytes.

3 - Se existir memória RAM em qualquer outro slot na região de memória que vai do endereço &H8000 até &HBFFF, esta não poderá ser usada.

4 - O endereço apontado pelos bytes &H8008 e &H8009 (início do texto BASIC), deverá conter o valor 00.

5 - Quanto aos números de linhas (para declarações que referenciam números de linhas, tal como GOTO, GOSUB, etc), a melhor opção é a transformação em Apontadores em Progresso, pois em se tratando de um cartucho de memória ROM eles nunca serão convertidos para apontadores quando em execução.

VARIÁVEIS DO SISTEMA DE SLOTS

Assim como outras variáveis do sistema, o MSX possui também uma área na qual estão contidas todas as informações (variáveis) a respeito dos slots, e que podem ser consultadas ou modificadas pelo sistema ou pelo usuário. Essas variáveis são atualizadas pelo sistema de modo que em qualquer momento o MSX possa saber o que está conectado em seus slots, como por exemplo, um cartucho com expansão de comandos (comando CALL), alguns dispositivos como um cartão de 80 colunas, RS232-C, etc.

Todas as variáveis do sistema que se referem aos slots são explicadas abaixo junto com os endereços onde estão localizadas cada uma delas.

EXPTBL - &HFCC1 - Indica qual slot está expandido. No caso de um slot estar expandido, ele deverá conter o valor &H80 e, caso contrário, deverá conter zero.

&HFCC1H	DEFB 00	PARA O SLOT 0
&HFCC2H	DEFB 00	PARA O SLOT 1
&HFCC3H	DEFB 00	PARA O SLOT 2
&HFCC4H	DEFB 00	PARA O SLOT 3

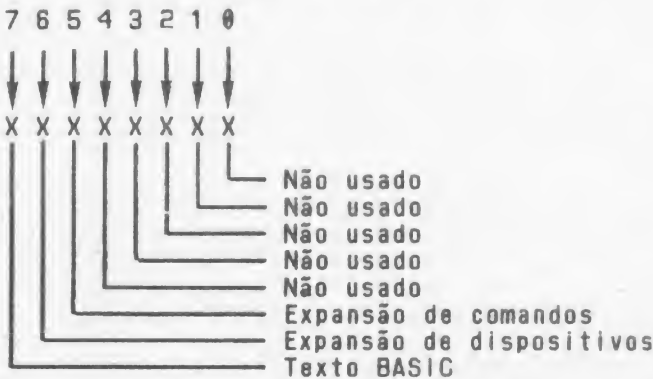
SLTTBL - &HFCC5 - Esta variável, que tem quatro bytes, indica qual o valor corrente na saída dos quatro possíveis registradores de slots secundários. O conteúdo de cada variável somente será válido se a referida variável em EXPTBL estiver com o valor &H80.

&HFCC5H	DEFB 00	PARA O SLOT 0
&HFCC6H	DEFB 00	PARA O SLOT 1
&HFCC7H	DEFB 00	PARA O SLOT 2
&HFCC8H	DEFB 00	PARA O SLOT 3

SLTATR - &HFCC9 - Esta variável do sistema que possui 64 bytes, contém os atributos para cada página, de todos os slots, inclusive os expandidos.

Cada byte nestas variáveis contém a informação correspondente a cada página, e o valor de cada um desses bytes deve estar de acordo com os referentes bits (setados ou resetados) conforme o seu significado, ilustrado na figura 5.3 .

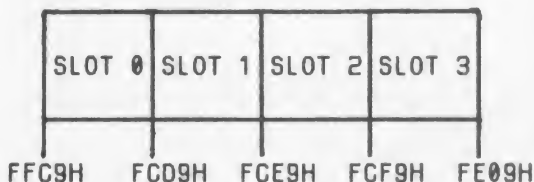
FIGURA 5.3 - Estrutura de um byte da SLTART



Entretanto, além de saber os valores destas variáveis, devemos saber como estão organizadas, e como localizar cada página de cada slot. Para isso, devemos levar em conta que :

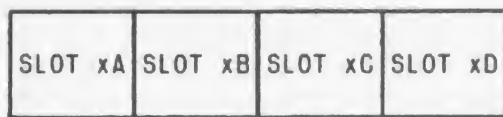
1 - Esses 64 bytes são divididos em quatro blocos de 16 bytes cada e abrangem inclusive os slots secundários, como mostrado na figura 5.4 .

FIGURA 5.4 - Divisão esquemática da SLTATR



2 - Entretanto cada slot mostrado na figura 5.4 tem mais quatro divisões, cada uma com 4 bytes que representam cada slot lógico, inclusive os expandidos, como mostrado na figura 5.5 .

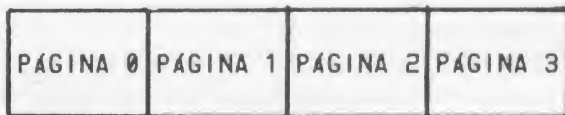
FIGURA 5.5 - Subdivisão da SLTATR.



O slot xA representa o slot primário x, onde x pode ser qualquer um dos slots de 0 a 3, o slot xB, o primeiro slot secundário, no caso de estar expandido, e assim por diante.

3 - Mas, cada slot da figura 5.5 é dividido em mais quatro partes, de 4 bytes cada, que representam as páginas de cada um dos slots, como ilustrado na figura 5.6

FIGURA 5.6 - Célula de 4 bytes da SLTATR.



SLTWRK - &HFD09 - Esta variável com o comprimento de 128 bytes, tem a mesma estrutura da variável **SLTATR**, mas, ao invés de ter apenas um byte para cada página, possui 2 bytes para indicar qual a área de trabalho reservada e/ou para servir como área de trabalho para cada página.

PROGNM - &HFD89 - Nesta variável de 16 bytes de comprimento, é colocado o nome para ser comparado. Veja no ítem 4 (**STATEMENT**) sobre o reconhecimento de cartuchos pelo sistema e no ítem 2 (**DEVICE**) sobre a expansão de dispositivos.

DEVICE - &HFD99 - Variável de identificação de um dispositivo, conforme a explicação sobre expansão de dispositivos.

INITGAMES - INICIALIZADORES DE JOGOS

Veremos neste ítem, o funcionamento dos inicializadores de jogos (**initgames**) que estavam em cartuchos e que funcionam nas páginas 1 e 2, de RAM, simultaneamente.

Os **initgames**, além de serem utilizados para fazer funcionar os jogos que estavam em cartuchos, também podem ser utilizados, isto é, quando bem utilizados, para relocar rotinas em linguagem de máquina para as páginas 0 e 1; transferir variáveis (por exemplo de um banco de dados) ou mesmo programas em **BASIC** para estas mesmas páginas e muitas outras aplicações.

Algumas dessas aplicações são mostradas a seguir, como por exemplo, os **initgames** de 16 Kbytes, de 32 Kbytes e também o uso do mesmo em conjunto com uma implementação do comando **CALL** através do **BASIC**.

INITGAME 16K

O programa da figura 5.7, foi feito para ser utilizado com programas de 16 Kbytes, que funcionem na página 1 do slot que tiver memória RAM, sendo que o mesmo deverá ser adicionado no final do programa **Assembly** com o qual vai ser usado.

Deve-se levar em conta que o **initgame** descrito abaixo irá fazer somente os testes de memória RAM nos slots primários, tendo em vista que no momento da edição deste livro ainda não haviam sido lançados tanto expensor de slots, como cartuchos de memória RAM.

FIGURA 5.7 - Initgame 16K

```

; &H9100
; ..
; .. PROGRAMA A SER DESLOCADO
; ..
; &HD0FF
;
;
D100 DBAB      ORG 0D100H
D102 C610      IN  A,(0ABH)      ;LE VALOR DA PPI.
                                ;SOMA 16 PARA OBTIER
                                ;O ULTIMO VALOR
D104 3248D1    LD  (VAR2),A      ;POSSIVEL PARA A PPI.
D107 C6F4      ADD  A,0F4H       ;SALVA O VALOR.
                                ;SOMA 244 PARA OBTIER
                                ;O PRIMEIRO VALOR
D109 3247D1    LD  (VAR1),A      ;POSSIVEL PARA A PPI.
D10C D3AB      OUT  (0ABH),A     ;SALVA O VALOR.
D10E           OUTRO:           ;MUDA A PPI.
D10E 3EAA      LD  A,0AAH
D110 320040    LD  (4000H),A      ;TESTA SE DA
D113 3A0040    LD  A,(4000H)     ;PARA ESCREVER
D116 FEAA      CP  0AAH          ;DADOS NA PAGINA 1.
D118 200C      JR  NZ,PROX       ;SE NAO DER, SALTA.
D11A 3E55      LD  A,055H       ;REPETE O TESTE COM
D11C 320040    LD  (04000H),A    ;OUTRO VALOR DE A.
D11F 3A0040    LD  A,(04000H)
D122 FE55      CP  055H
D124 2812      JR  Z,TRANSF
D126           PROX:
D126 3A47D1    LD  A,(VAR1)
D129 C604      ADD  A,04H
D12B 3247D1    LD  (VAR1),A
D12E D3AB      OUT  (0ABH),A
D130 47        LD  B,A
D131 3A48D1    LD  A,(VAR2)
D134 B8        CP  B
D135 20D7      JR  NZ,OUTRO
D137 C9        RET
D138           TRANSF:
D138 210080    LD  HL,8000H
D13B 110040    LD  DE,4000H
D13E 010040    LD  BC,4000H
D141 ED80      LDIR
D143 2A0240    LD  HL,(4002H)
D146 E9        JP  (HL)
D147 00        VAR1= DEFB 0
D148 00        VAR2= DEFB 0
D149           END

```

Na figura 5.7, a rotina que realmente nos interessa é justamente aquela que faz os testes para a localização da memória RAM (endereço &HC000 até &HC037) pois essa mesma rotina pode ser utilizada com qualquer programa, desde que sejam feitas as devidas modificações.

Note que o programa a ser relocado deve ficar a partir do endereço &H9100, pois assim existe a possibilidade de ser feito um programa de apresentação em BASIC e que também carregue o JOGO.

INITGAME 32K

Esta versão de initgame tem, como os initgames anteriores, a mesma rotina principal de teste, mas, só que desta vez será utilizada para relocar um programa de 32K usando as páginas 1 e 2 do slot que tiver memória RAM.

Será usado um programa em BASIC para carregar as duas partes do programa de 32K. Ele será destruído durante o carregamento da segunda, pois a área normal de execução do programa em assembly é a ocupada pelo programa BASIC.

Observe na figura 5.8 alguns exemplos de programas garregadores em BASIC.

FIGURA 5.8 - Exemplos de programas

```
10 PRINT"Carregando jogo xxxxxxxxxxxx"  
20 BLOAD"JOGO-A.BIN",R  
30 BLOAD"JOGO-B.BIN",R
```

Ou qualquer outro programa que tenha como efeito o carregamento de programas em Assembly. O primeiro initgame de 32K está ilustrado na figura 5.9.

FIGURA 5.9 - Initgame 32K (1ª PARTE)

```

?
?
?
?9100
? - -
? - -   PROGRAMA NO. 1
? - -
?D0FF
?
?
      ORG    @D100H
D100  DBAB      IN     A,(0ABH)
D102  324CD1    LD     (VAR1),A
D105  C610      ADD    A,10H
D107  324DD1    LD     (VAR2),A
D10A  C6F4      ADD    A,0F4H
D10C  324CD1    LD     (VAR1),A
D10F  D3AB      OUT    (0ABH),A
D111  3EAA      OUTRO: LD     A,0AAH
D113  320040    LD     (4000H),A
D116  3A0040    LD     A,(4000H)
D119  FEAA      CP     0AAH
D11B  200C      JR     NZ,PROX
D11D  3E55      LD     A,055H
D11F  320040    LD     (04000H),A
D122  3A0040    LD     A,(04000H)
D125  FE55      CP     055H
D127  2812      JR     Z,TRANSF

```

```

D12Y 3A4CD1    PROX: LD    A, (VAR1)
D12C C604      ADD    A, 04H
D12E 324CD1    LD     (VAR1), A
D131 D3AB      OUT    (0ABH), A
D133 47        LD     B, A
D134 3A4DD1    LD     A, (VAR2)
D137 88        CP     B
D138 20D7      JR     NZ, OUTRO
D13A C9        RET
D13B 210091    TRANSF: LD    HL, 9100H
D13E 110040    LD     DE, 4000H
D141 010040    LD     BC, 4000H
D144 EDB0      LDIR
D146 3A4CD1    LD     A, (VAR1)
D149 D3AB      OUT    (0ABH), A
D14B C9        RET
D14C 00        VAR1: DEFB 0
D14D 00        VAR2: DEFB 0
D14E           END

```

Como podemos observar, nesta pequena rotina de teste foram incluídas mais três instruções que armazenam e recuperam um byte na região de dados que nos fornece o estado original da configuração dos slots. Este dado só será usado para reconfigurar os slots quando o programa for voltar ao BASIC.

Ao contrário dos initgames anteriores, neste devemos relocar a primeira parte do programa Assembly para a página 1. Depois retornar ao BASIC, prosseguindo com o carregamento da segunda parte.

A segunda parte deve ser carregada a partir do endereço &H8000, pois não precisaremos mais do programa carregador BASIC.

A diferença entre o initgame do primeiro e do segundo blocos, é que no segundo não há as instruções de transferência de bloco (LDIR), pois o segundo bloco já é carregado na sua área de execução. Ele também habilita a página 1 que já contém o primeiro bloco, pesquisa os bytes que contém o endereço de execução (INIT) e parte para a execução do cartucho.

Para que possa ficar mais claro, examine a rotina final do programa na figura 5.10.

FIGURA 5.10 - Initgame 32K (2ª PARTE)

```

;
;
;B000
; --
; --  PROGRAMA NO. 2
; --
;BFFF
;
;
;          ORG  0C000H
C000 DBAB      IN    A, (0ABH)
C002 3240C0    LD     (VAR1), A
C005 C610      ADD    A, 10H
C007 3241C0    LD     (VAR2), A
C00A C6F4      ADD    A, 0F4H
C00C 3240C0    LD     (VAR1), A
C00F D3AB      OUT    (0ABH), A
C011          OUTRO:
C011 3A40C0    LD     A, (VAR1)

```

```

C014 320040      LD      (4000H),A
C017 3A0040      LD      A,(4000H)
C01A FEAA        CP      0AAH
C01C 200C        JR      NZ,PROX
C01E 3E55        LD      A,055H
C020 320040      LD      (04000H),A
C023 3A0040      LD      A,(04000H)
C026 FE55        CP      055H
C028 2812        JR      Z,EXECUT
C02A              PROX:
C02A 3A40C0      LD      A,(VAR1)
C02D C604        ADD     A,04H
C02F 3240C0      LD      (VAR1),A
C032 D3AB        OUT     (0ABH),A
C034 47          LD      B,A
C035 3A41C0      LD      A,(VAR2)
C038 B8          CP      B
C039 20D6        JR      NZ,OUTRO
C03B C9          RET
C03C              EXECUT:
C03C 2A0240      LD      HL,(4002H)
C03F E9          JP      (HL)
C040 00          VAR1: DEFB 0
C041 00          VAR2: DEFB 0
C042              END

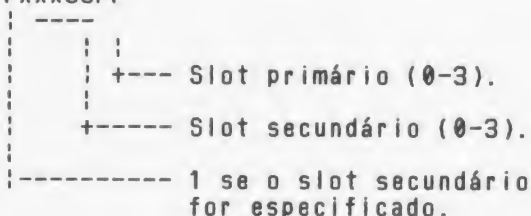
```

INITGAME PLUS

Nesse initgame, não serão usadas as rotinas "normais" de transferência de blocos, mas duas rotinas existentes no BIOS.

A primeira rotina é a WRSLT, que tem como entry point o endereço &H0014 e necessita dos seguintes valores nos registradores:

A - FxxxSSPP



HL - Endereço de memória no qual será escrito o dado.

E - Dado a ser escrito.

A segunda rotina é a CALLF, tem como entry point o endereço &H0030, e ela seleciona o slot apropriado para a execução automaticamente. Sua execução é feita através da instrução RST 30H da seguinte forma:

```

RST 30H
DEFB xx      ;Slot em que está o programa a
              ser executado.
DEFW xxxx    ;Endereço de execução do programa

```

Esta versão de initgame é dada na figura 5.11

FIGURA 5.11 - Initgame usando rotinas do BIOS

```

; 8000
; --
; -- PROGRAMA A SER DESLOCADO
; --
; BFFF
;

ORG 0C000H
IN A,(0ABH) ;LE O VALOR DA PPI.
AND 11000000B
RLC A
RLC A
LD (0C01DH),A ;ALTERA O PROGRAMA
LD (0C02CH),A ;COM O VALOR CORRETO
;DA PPI PARA O SLOT.
C00E 010040 LD BC,4000H ;NUMERO DE BYTES.
C011 110080 LD DE,8000H ;END. ORIGEM.
C014 210040 LD HL,4000H ;END. DESTINO.
C017
PROX:
C017 C5 PUSH BC ;SALVA REGISTROS.
C018 E5 PUSH HL
C019 D5 PUSH DE
C01A 1A LD A,(DE) ;LE DADO A TRANSFERIR.
C018 5F LD E,A
C01C 3E00 LD A,00 ;A CONTEM SLOT DESTINO.
C01E CD1400 CALL 0014H ;COPIA O DADO.
C021 D1 POP DE
C022 E1 POP HL ;RESTAURA REGISTROA.
C023 C1 POP BC
C024 23 INC HL
C025 13 INC DE ;ATUALIZA VALORES.
C026 0B DEC BC
C027 78 LD A,B ;TESTA SE CHEGOU A 0.
C028 B1 OR C ;SE NAO, VOLTA
C029 20EC JR NZ,PROX ;AO LOOP.
C02B F7 RST 30H ;FAZ A CHAMADA.
C02C 00 DEFB 00 ;NUMERO DO SLOT.
C02D 0000 DEFW 0000 ;ENDEREIO A CHAMAR.

```

Esse initgame, só reloca e executa um programa de 16K. Algumas modificações podem ser feitas facilmente para adaptá-lo ao tamanho do programa a ser deslocado.

O uso de rotinas do BIOS limitam o initgame em alguns pontos. O endereço de execução do programa tem que ser especificado no próprio initgame (devido ao RST 30H), só trabalhar com os computadores com 64 Kbytes de RAM e testar somente os slots primários.

IMPLEMENTANDO O COMANDO CALL

Vamos apresentar, nesse ítem, um programa simples mostrando a implementação do comando CALL através do BASIC, utilizando o initgame plus e alterando algumas variáveis de sistema.

Supondo que não exista espaço de memória suficiente para trabalhar com um programa BASIC e com rotinas em linguagem de máquina. Uma possível solução poderia ser a transformação de uma boa parte do programa BASIC em linguagem de máquina. Infelizmente na

maioria dos casos isto não é muito fácil. Outra solução seria a compactação do programa BASIC ao máximo possível.

Caso não seja possível realizar nenhuma dessas opções, poderemos "enganar" o sistema para que o mesmo reconheça algumas implementações (ou rotinas em Assembly) como se estivessem em cartucho.

Para tal, deveremos escrever o programa em Assembly; como se o mesmo estivesse em cartucho, adicionarmos ao mesmo um initgame e atualizar o valor da variável de sistema SLTATR de acordo com a implementação que, neste caso será a do comando CALL. Devido à implementação ser do comando CALL, o valor em SLTATR será de 20H.

Entretanto, deve ser lembrado que o sistema MSX não reconhece por si só um dado nome utilizado em conjunto com o comando CALL e é necessário realizar esse teste mesmo que só vá ser utilizado um só nome. Caso contrário, poderemos colocar em conflito, várias chamadas (caso existam) desse comando.

Para que possa ser melhor compreendido, o programa que realiza essa implementação foi dividido em duas partes.

Na primeira parte do programa, a rotina da figura 5.12 irá fazer apenas o teste de slots, deslocar a rotina para a página 1 e atualizar a variável SLTATR de acordo com a página em que irá funcionar.

FIGURA 5.12 - Inicialização da implementação de um comando

```

;
;
; 8000
; --
; -- PROGRAMA A SER DESLOCADO
; --
; BFFF
;

C000 DBAB          ORG 0C000H
C002 E6C0          IN  A,(0ABH)          ;LE O VALOR DA PPI.
C004 CB07          AND 11000000B
C006 CB07          RLC A
C008 321DC0        LD  (0C01DH),A
C00B 322CC0        LD  (0C02CH),A          ;ALTERA O PROGRAMA
                                          ;COM O VALOR CORRETO
                                          ;DA PPI PARA O SLOT.
C00E 010040        LD  BC,4000H          ;NUMERO DE BYTES.
C011 110080        LD  DE,8000H          ;END. ORIGEM.
C014 210040        LD  HL,4000H          ;END. DESTINO.
C017
C017 C5            PUSH BC
C018 E5            PUSH HL
C019 D5            PUSH DE
C01A 1A            LD  A,(DE)
C01B 5F            LD  E,A
C01C 3E00          LD  A,00
C01E CD1100        CALL 0014H
C021 D1            POP  DE
C022 E1            POP  HL
;SALVA REGISTROS.
;LE DADO A TRANSFERIR.
;A CONTEM SLOT DESTINO.
;COPIA O DADO.
;
;RESTAURA REGISTROA.

```


C023 C1	POP BC	
C024 23	INC HL	
C025 13	INC DE	;ATUALIZA VALORES.
C026 08	DEC BC	
C027 78	LD A,B	;TESTA SE CHEGOU A 0.
C028 B1	OR C	;SE NAO, VOLTA
C029 20EC	JR NZ,PROX	;AO LOOP.
C028 F7	RST 30H	;FAZ A CHAMADA.
C02C 00	DEFB 00	;NUMERO DO SLOT.
C02D 0000	DEFW 0000	;ENDEREIO A CHAMAR.

Os bytes entre o endereço &H9065 até &H9090 nada mais são que o initgame plus e portanto não há necessidade de comentá-los.

Do endereço &H9091 até &H90A4 é onde, realmente, se encontra a rotina de procura pelo slot de memória RAM. A multiplicação por 16 serve para localizar qual dos 64 bytes da SLTATR deverá ser atualizado e o valor 1 é somado para fornecer à rotina, a página em que será usada a extensão de comandos.

FIGURA 5.13 - Programa a ser chamado pelo CALL

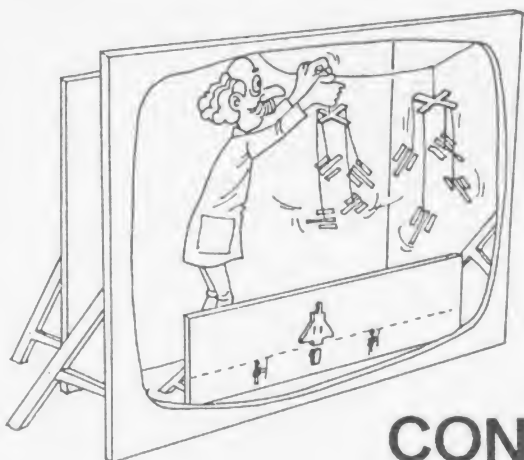
9000 41420000	ORG 09000H	
9004 10400000	DEFW 4241H,0	;IDENTIF. CARTUCHO.
9008 00000000	DEFW 4010H,0	;ENDEREIO DO CALL.
900C 00000000	DEFW 0,0	
9010 E5	DEFW 0,0	
9011 2189FD	PUSH HL	;SALVA POINTER DO BASIC.
9014 114040	LD HL,0FD89H	;HL CONTEM PROCNM.
9017 0606	LD DE,4040H	;DE COM O NOME A TESTAR.
	LD B,06	;B COM O TAMANHO
	;DO NOME + 1.	
9019	PROX:	
9019 1A	LD A,(DE)	;COMPARA LETRA A LETRA
901A BE	CP (HL)	;DOS DOIS NOMES.
901B 2006	JR NZ,FIM	;SE HA DIFERENCA SALTA
	;PARA O 'Syntax error'.	
901D 23	INC HL	
901E 13	INC DE	;PROXIMO BYTE.
901F 10FB	DJNZ PROX	
9021 1803	JR PRINT	
9023	FIM:	
9023 E1	POP HL	;RESTAURA POINTER.
9024 37	SCF	;SETA A CARRY FLAG
	*PARA IMPRIMIR A	
	;MENSAGEM DE ERRO.	
9025 C9	RET	
9026	PRINT:	
9026 CDC300	CALL 00C3H	;LIMPA A TELA.
9029 3E0A	LD A,0AH	;POSICIONA O CURSOR
902B 320CF3	LD (0F3DCH),A	;NA POSICAO 10,10.
902E 32DDF3	LD (0F3DDH),A	
9031 214640	LD HL,4046H	;INICIO DA MENSAGEM.
9034 110000	LD DE,0000H	;INICIO DA URAM.
9037 011800	LD BC,0018H	;TAMANHO DA MENSAGEM.
903A CD5C00	CALL 005CH	;IMPRIME.
903D AF	XOR A	;ZERA A CARRY FLAG
	;P/ NAO ACUSAR ERRO.	
903E E1	POP HL	;RESTAURA POINTER.
903F C9	RET	;RETORNA.
9040 54455354	DEFW 'TESTE'	
9044 45		
9045 00	DEFB 0	
9046 50524F47	DEFW 'PROGRAMACAO AVANÇADA NO MSX'	

```
904A 52414D41
904E 43414F20
9052 4156414E
9056 43414441
905A 204E4F20
905E 4D5358
9061 00000000
```

```
DEFW 0,0
```

No programa da figura 5.13 , temos entre os endereços &H9000 e &H9022, a rotina que realiza o teste para verificar se o comando CALL TESTE é válido. Caso o mesmo seja, o programa continuará no endereço &H9026 (ou &H4026, quando em sua posição real), realizando um CLS e imprimindo a mensagem " PROGRAMAÇÃO AVANÇADA NO MSX" e retornando ao BASIC com a carry flag resetada para que o sistema reconheça a aceitação do comando. Se a palavra TESTE não for usada, o programa setará a CARRY FLAG e o sistema imprimirá uma mensagem de erro.

Deve-se levar em consideração, que o MSX quando da entrada de uma palavra através do comando CALL, transforma todas as letras minúsculas em maiúsculas, portanto, a palavra que estiver no programa também deverá ser maiúscula, para que haja uma perfeita comparação.



CONSTRUINDO UM JOGO EM ASSEMBLY

Chegamos, finalmente, ao último capítulo em que mostraremos um exemplo prático de muito do que foi visto ao longo deste livro. Veremos a construção de um jogo do tipo "videogame", com naves espaciais, bombas, invasores e lasers, bem ao estilo dos jogos "SPACE INVADERS", chamado ORPHEUS. Neste jogo você controla um canhão terrestre que deve defender a Terra contra os invasores que entram soltando bombas. Em cada fase há oito invasores que atacam. Oito fases constituem um estágio que, sendo vencido, torna-se mais difícil devido à ação de sabotadores subterrâneos que tentam acertar seu canhão com um desintegrador de matéria.

Vamos ver, então, algumas estruturas do programa.

Começamos pelo acesso aos "arrays", muito importante que foi usado repetidas vezes durante o jogo. Por esse método, o número do elemento é dado no registrador A, seu endereço retorna no par HL e o conteúdo volta em A. A figura 6.1 apresenta a listagem do acesso aos arrays.

O valor ARRAY aponta o endereço inicial do "array" (elemento de índice zero). Neste caso cada elemento ocupa um byte, mas pode ocupar n bytes, bastando para isso repetir a instrução ADD HL,B6 e LD A,(HL) n vezes.

FIGURA 6.1 - Controle dos invasores

```

#
LD HL,ARRAY
LD B,0
LD C,A
ADD HL,BC
LD A,(HL)
RET
ARRAY: DB 2,3,1,10,2,4...

```

Outra novidade é o pseudo-aleatorizador, que sempre devolve o valor entre 0 e 255 no registro A. Na verdade, ele apenas lê bytes sucessivos de trechos "estratégicos" da ROM que se parecem muito com séries aleatórias. Uma limitação dessa rotina é a sua faixa de valores, estritamente fixada entre 0 e 255. Como muitos trechos do jogo não funcionariam para essa faixa, adotou-se um truque que consiste em truncar os bits mais altos do valor retornado, passando a assumir um novo valor menor que o anterior. Para fazer isto usa-se a instrução AND n, com n valendo 3,7,15, etc para truncar os bits altos do registro A.

Devido à impossibilidade de se incluir os comentários na listagem do ORPHEUS devido à falta de memória, aqui estão eles em anexo:

ENDEREÇO COMENTÁRIO

C350 a C363:apaga os sprites dos sabotadores da tela.
 C364 a C36F:imprime o cabeçalho do jogo.
 C370 a C387:inicializa as variáveis.
 C388 a C393:define os atributos de sprites do canhão.
 C394 a C3A5:acerta a posição do canhão e dos tiros.
 C3A6 a C3B3:inicialização final do canhão e bombas.
 C3B4 a C3B6:inicialização das fases.
 C3B7 a C3C3:zera os contadores de invasores destruídos, pausa de tempo e "status" do canhão.
 C3C4 a C3D2:imprime o número das naves restantes.
 C3D3 a C410:controla o canhão do jogador.
 C411 a C462:controla os disparos dos tiros.
 C463 a C46B:chama os elementos móveis do jogo.
 C46C a C472:testa a explosão do canhão.
 C473 a C4A4:movimenta os tiros.
 C4A5 a C4BE:coordena invasores, bombas e verifica colisão.
 C4BF a C4E4:controla o som do tiro.
 C4E5 a C4EE:testa se todos os invasores foram destruídos e encerra loop central.
 C4EF a C4F4:procede troca da fase.

C4F5 a C511:aleatorizador.
 C512 a C52C:testa pressionamento do botão de tiro.
 C52D a C553:movimenta os tiros.
 C554 a C5AC:imprime a nave.
 C5AD a C5C1:verifica se o invasor existe.
 C5C2 a C55C:calcula nova direção do invasor.
 C55D a C621:movimenta e imprime o invasor.
 C622 a C626:verifica se o invasor está em movimento.
 Se estiver, chama a rotina de bombas.
 C627 a C647:encerra loop dos invasores e atualiza
 temporização.
 C648 a C654: copia as coordenadas dos invasores na
 VRAM.
 C655 a C669:reinicia as trajetórias dos invasores.
 C670 a C6C8:movimenta as estrelas.
 C6C9 a C6D6:temporiza os sabotadores.
 C6D7 a C6DC:verifica se não há sabotadores a imprimir.
 C6DD a C740:movimenta os sabotadores.
 C741 a C765:ataca o canhão por baixo.
 C766 a C789:faz a explosão do canhão.
 C790 a C796:sorteia se haverá nova bomba.
 C797 a C7A3:calcula um índice e verifica sua validade.
 C7A4 a C7B3:verifica se a bomba desse índice existe.
 C7B4 a C7C2:acerta a posição da bomba.
 C7C3 a C7D7:programa a direção da bomba.
 C7D8 a C818:atualiza a posição da bomba.
 C819 a C825: copia as coordenadas das bombas na VRAM.
 C826 a C844:inicializa as bombas.
 C845 a C84A:testa se houve choque entre sprites.
 C84B a C892:coordena o teste de tiros.
 C893 a C8A5:coordena o teste de bombas e invasores.
 C8A6 a C8DF:comparador comum bomba/invasor.
 C8E0 a C929:sub-rotina de comparação do tiro com
 invasor.
 C92A a C94C:atualiza contador de invasores destruídos
 e controla o placar.
 C94D a C983:inicializa os invasores.
 C984 a C991:acerta os registros do PSG.
 C992 a C9D8:imprime os placares.
 C9D8 a CA2C:inicia os sabotadores, bombas, invaso-
 res, e contadores de pontos e invasores.

DIGITANDO O PROGRAMA ORPHEUS

Se você deu uma olhada no Apêndice IV, já é possível digitar o programa ORPHEUS. Ele é formado por quatro listagens: uma em BASIC, que inicializa e controla o jogo e três blocos de dados: a primeira formata a tela, outra executa o jogo e a terceira define as trajetórias dos invasores.

Assim, comece digitando a figura 6.2 e grave-a com o nome "ORPHEUS".

FIGURA 6.2 - Controlador do ORPHEUS, em BASIC

```
10 DATA 0,00,00,01,01,01,03,07,0D,1D,3D,
7D,7D,7F,01,00,00,80,80,C0,C0,C0,E0,F0,D
8,DC,DE,DF,DF,FF,C0,00,00
20 DATA 1,00,00,00,01,01,01,00,00,00,00,
00,00,00,00,00,00,00,80,C0,80,C0,C0,8
0,80,80,00,00,00,00,00,00
30 DATA 2,00,00,00,01,01,00,00,00,00,00,
00,00,00,00,00,00,00,80,80,40,80,80,80,0
0,00,00,00,00,00,00,00,00
40 DATA 3,00,00,00,00,00,00,00,00,02,02,02,
02,02,00,00,00,00,00,00,00,00,00,00,00,2
0,20,20,20,20,00,00,00,00
50 DATA 4,00,00,00,00,00,00,00,00,00,00,00,
00,00,00,00,00,00,80,80,80,80,80,00,00,0
0,00,00,00,00,00,00,00,00
60 DATA 5,00,00,00,01,01,00,00,00,00,00,
00,00,00,00,00,00,80,80,80,C0,C0,80,00,0
0,00,00,00,00,00,00,00,00
70 DATA 6,00,00,01,03,03,01,07,0B,07,03,
02,02,02,0E,00,00,00,00,20,A0,A0,20,E0,A
0,E0,A0,A0,80,80,80,E0,00
80 DATA 7,00,00,01,03,03,01,07,0B,07,03,
02,02,02,02,0E,00,00,00,20,A0,A0,20,E0,A
0,E0,A0,A0,80,80,E0,00,00
90 DATA 8,00,00,01,03,03,01,07,07,03,03,
02,02,02,02,0E,00,20,20,20,A0,A0,20,E0,E
0,A0,80,80,80,80,80,E0,00
100 DATA 9,00,00,00,00,00,01,09,1D,1F,1D
,1F,1D,1F,1C,1C,08,00,00,80,80,80,C0,C8,
DC,FC,DC,FC,DC,FC,9C,1C,08
110 DATA 10,00,00,00,00,00,00,01,03,3F,7
F,1C,1D,03,07,06,00,00,00,04,08,10,60,C0
,C8,30,70,E0,C0,80,00,00,00
120 DATA 11,00,00,00,3F,7F,3F,15,1F,3F,1
F,15,3F,7F,3F,00,00,00,00,80,C0,80,00
,E0,FC,E0,00,80,C0,80,00,00
```

```

130 DATA 12,00,01,01,07,07,37,39,1D,0E,0
7,03,01,00,00,00,00,00,80,80,80,80,C0
,C0,60,70,B0,88,44,02,00,00
140 DATA 13,00,10,38,3F,3B,3F,3B,3F,3B,1
3,03,01,01,01,00,00,00,10,38,F8,B8,F8,B8
,F8,B8,90,80,00,00,00,00,00
150 DATA 14,00,00,00,00,00,00,01,01,03,0
7,06,08,11,20,00,00,00,40,C0,F0,F0,F6,CE
,DC,38,70,E0,C0,00,00,00,00
160 DATA 15,00,00,01,03,01,00,07,3F,07,0
0,01,03,01,00,00,00,00,00,FC,FE,FC,A8,F8
,FC,F8,A8,FC,FE,FC,00,00,00
170 DATA 16,00,00,20,10,08,06,07,13,0C,0
E,07,03,01,00,00,00,00,00,00,00,00,00,00
,C0,FC,FE,38,B8,C0,E0,60,00
180 DATA 17,00,1C,32,4D,56,A8,DD,7E,2B,7
5,6A,A8,B6,49,67,1E,10,7C,CE,23,95,2A,94
,4A,41,A9,12,6C,FA,E1,4B,36
190 SCREEN1,2,0:KEYOFF:WIDTH32:COLOR 15,
1,1:LOCATE 3,10:PRINT "Aguarde a carga d
os blocos":FOR I=0 TO 17:A$="":READ N:F0
R J=0 TO 31:READ B$:A$=A$+CHR$(VAL("&H"+
B$)):NEXT J:SPRITE$(N)=A$:NEXT I
200 BLOAD "JOGO":BLOAD "TELA":BLOAD "TRA
J"
210 FOR F=0 TO 128:VPOKE F,0:NEXT F:VPOK
E 0,128:VPOKE 64,128:VPOKE8192,113:VPOKE
8193,129:VPOKE8216,33:DEFUSR=&H9000:DEFU
SR1=&HC350:DEFUSR2=&H156
220 X=USR(0):LOCATE 0,22:PRINT STRING$(3
2,196);:LOCATE 10,11:PRINT "TECLE ESPAÇO
"
230 IF INKEY$("<")="" THEN 230 ELSE X=USR(0
):LOCATE 0,22:PRINT STRING$(32,196);:X=U
SR1(0)+USR2(0):GOTO 220

```

Se você não possui um compilador, carregue o Monitor e digite M &H9000. A seguir, entre com os bytes da figura 6.3 ou digite o programa fonte em Assembly, de acordo com a sintaxe do seu compilador e, após terminar, salve-a com

BSAVE "TELA",&H9000,&H902F

FIGURA 6.3 - Rotina formata-tela em Assembly

```

                                ORG 09000H
004D WRTVRM: EQU 04DH
9000 210071 LD HL,7100H
9003 AF XOR A
9004
9004 E5 LOOPC: PUSH HL ;SALVA HL.
9005 56 LD D,(HL) ;LE CODIGO INICIAL
                                ;DA COLUNA.
                                ;CALCULA ENDEREÇO
                                ;DA VRAM.
9006 210018 LD HL,6144
9009 0600 LD B,0
900B 4F LD C,A
900C 09 ADD HL,B
900D F5 PUSH AF
900E 5F LD E,A
900F AF XOR A
9010
9010 F5 LOOPL: PUSH AF
9011 7A LD A,D
9012 E607 AND 7 ;REDUZ CODIGO
                                ;AO MODULO 8.
                                ;TESTA SE A COLUNA
                                ;E PAR.
                                ;SE FOR, SOMA 8
                                ;AO COD. CARACTERE.
                                ;ESCREVE NA TELA.
                                ;INCREMENTA CODIGO.
9014 C843 BIT 0,E
9016 2802 JR Z,BIT0
9018 C608 ADD A,8
901A
901A CD4D00 BIT0: CALL WRTVRM
901D 14 INC D
901E 0E20 LD C,32
9020 09 ADD HL,B
9021 F1 POP AF ;DESCE 1 LINHA.
9022 3C INC A
9023 FE18 CP 24
9025 20E9 JR NZ,LOOPL ;FECHA LOOP LINHAS.
9027 F1 POP AF
9028 E1 POP HL ;RESTAURA REGISTROS.
9029 23 INC HL ;PROXIMA COLUNA.
902A 3C INC A
902B FE20 CP 32
902D 20D5 JR NZ,LOOPC ;FECHA LOOP COLUMNAS.
902F C9 RET

```

O mesmo deve ser feito com a figura 6.4 : voltando ao monitor, digite M &HC350 e entre os dados pelo monitor, ou compile o programa fonte de acordo com a sintaxe do seu compilador. Após terminar, grave-o com

BSAVE "JOGO",&HC350,&HCB57

FIGURA 6.4 - Jogo ORPHEUS, em Assembly

```

                                ORG 50000
;----- BLOCO 1:CONSTANTES
004A RDVRM: EQU 04AH
004D WRTVRM: EQU 04DH
005C LDIRVM: EQU 05CH
0073 WRTPSG: EQU 073H
00D5 GTSTCK: EQU 0D5H
00D8 GTTRIG: EQU 0D8H
013E RDVDP: EQU 13EH
0200 INVRAM: EQU 200H
0003 TCAN: EQU 003H
0008 LIINV: EQU 008H
00B0 PAUSA: EQU 0B0H
0008 COMPU: EQU 008H
0002 TPINV: EQU 002H
A000 IPVR: EQU 0A000H
;----- BLOCO 2:INICIA JOGO
INICIO:
C350
C350 3ED1 LD A,209
C352 21541B LD HL,6996
C355 CD4D00 CALL WRTVRM
C358 21581B LD HL,7000
C35B CD4D00 CALL WRTVRM

```

```

C35E 215C1B      LD      HL,7004
C361 CD4D00      CALL   WRTVRM
C364 2136CB      LD      HL,MSTEL
C367 11001B      LD      DE,6144
C36A 012000      LD      BC,32
C36D CD5C00      CALL   LDIRVM
C370 3E03        LD      A,3
C372 3249CA      LD      (NAVES),A
C375 210000      LD      HL,0
C378 2262CA      LD      (PONTOS),HL
C37B CD92C9      CALL   PLACAR
C37E AF          XOR     A
C37F 324DCA      LD      (LISAB),A
C382 323ECA      LD      (FASE),A
C385 CDD8C9      CALL   STFASE

;----- BLOCO 3:INICIA NAVE
INNAVE:
C388 21DECA      LD      HL,ARPC
C38B 11001B      LD      DE,6912
C38E 011400      LD      BC,20
C391 CD5C00      CALL   LDIRVM
C394 3EA0        LD      A,160
C396 322ECA      LD      (YNAVE),A
C399 3E80        LD      A,128
C39B 322DCA      LD      (XNAVE),A
C39E 3ED1        LD      A,209
C3A0 3230CA      LD      (YTIRO),A
C3A3 3232CA      LD      (YTIRO2),A
C3A6 3E03        LD      A,TCAN
C3A8 3238CA      LD      (CCAN),A
C3AB 210002      LD      HL,512
C3AE 2260CA      LD      (FTIRO),HL
C3B1 CD26CB      CALL   INIBMB

;----- BLOCO 4:INICIA FASE
INFASE:
C3B4 324DC9      CALL   ININV
C3B7 AF          XOR     A
C3B8 3238CA      LD      (CONTOS),A
C3BB 3242CA      LD      (PSINC),A
C3BE 3243CA      LD      (CCINV),A
C3C1 3235CA      LD      (STATX),A
C3C4 3E1E        LD      A,30
C3C6 D399        OUT     (153),A
C3C8 3E18        LD      A,24
C3CA D399        OUT     (153),A
C3CC 3A49CA      LD      A,(NAVES)
C3CF C62F        ADD     A,47
C3D1 D398        OUT     (152),A

;----- BLOCO 5:LOOP CENTRAL
CENTRAL
C3D3 3A3BCA      LD      A,(CCAN)
C3D6 3D          DEC     A
C3D7 3238CA      LD      (CCAN),A
C3DA A7          AND     A
C3DB 2034        JR      NZ,SEGUE
C3DD 3E03        LD      A,TCAN
C3E1 3238CA      LD      (CCAN),A
C3E2 3A37CA      LD      A,(MODE)
C3E5 CDD500      CALL   GTSTCK
C3E8 FE03        CP      3
C3EA 280B        JR      Z,DIR
C3EC FE07        CP      7
C3EE 2814        JR      Z,ESQ
C3F0 FE01        CP      1
C3F2 FE05        CP      5
C3F4 C8          RET     Z
C3F5 181A        JR      SEGUE
C3F7

DIR:
C3F7 3A2DCA      LD      A,(XNAVE)
C3FA FEEA        CP      234
C3FC 3013        JR      NC,SEGUE
C3FE 3C          INC     A
C3FF 322DCA      LD      (XNAVE),A
C402 180D        JR      SEGUE
C404

ESQ:
C404 3A2DCA      LD      A,(XNAVE)

```

C407	FE10	CP	16
C409	3806	JR	C, SEGUE
C40B	3D	DEC	A
C40C	322DCA	LD	(XNAVE), A
C40F	1800	JR	SEGUE
C411		SEGUE:	
C411	3A37CA	LD	A, (MODE)
C414	CD12C5	CALL	TRIGGER
C417	FEFF	CP	255
C419	204B	JR	NZ, SEGUE2
C41B	3A30CA	LD	A, (YTIRO)
C41E	FED1	CP	209
C420	201B	JR	NZ, TIRO2
C422	21061B	LD	HL, 691B
C425	3E10	LD	A, 16
C427	CD4D00	CALL	WRTVRH
C42A	3A2ECA	LD	A, (YNAVE)
C42D	D60B	SUB	B
C42F	3230CA	LD	(YTIRO), A
C432	3A2DCA	LD	A, (XNAVE)
C435	322FCA	LD	(XTIRO), A
C43B	1B1D	JR	STIRO
C43A		TIRO2:	
C43A	3A32CA	LD	A, (YTIRO2)
C43D	FED1	CP	209
C43F	2022	JR	NZ, SEGUE2
C441	210A1B	LD	HL, 6922
C444	3E10	LD	A, 16
C446	CD4D00	CALL	WRTVRH
C449	3A2ECA	LD	A, (YNAVE)
C44C	D60B	SUB	B
C44E	3232CA	LD	(YTIRO2), A
C451	3A2DCA	LD	A, (XNAVE)
C454	3231CA	LD	(XTIRO2), A
C457		STIRO:	
C457	2122CB	LD	HL, SDTIRO
C45A	CD84C9	CALL	STSND
C45D	216400	LD	HL, 100
C460	2260CA	LD	(FTIRO), HL
C463		SEGUE2:	
C463	CD54C5	CALL	IMPNAV
C466	CD70C6	CALL	ESTRTM
C469	CD09C6	CALL	SABOT
C46C	3A35CA	LD	A, (STATX)
C46F	A7	AND	A
C470	C266C7	JP	NZ, EXPLS
C473	3A30CA	LD	A, (YTIRO)
C476	FED1	CP	209
C47B	2B12	JR	Z, TIROX2
C47A	3234CA	LD	(YTR), A
C47D	3A2FCA	LD	A, (XTIRO)
C480	3233CA	LD	(XTR), A
C483	CD2DC5	CALL	TIROS
C486	3A34CA	LD	A, (YTR)
C489	3230CA	LD	(YTIRO), A
C48C		TIROX2:	
C48C	3A32CA	LD	A, (YTIRO2)
C48F	FED1	CP	209
C491	2B12	JR	Z, SEGUE3
C493	3234CA	LD	(YTR), A
C496	3A31CA	LD	A, (XTIRO2)
C499	3233CA	LD	(XTR), A
C49C	CD32C5	CALL	TIROS2
C49F	3A34CA	LD	A, (YTR)
C4A2	3232CA	LD	(YTIRO2), A
C4A5		SEGUE3:	
C4A5	A7	AND	A
C4A6	A7	AND	A
C4A7	3A43CA	LD	A, (CCINV)
C4AA	3C	INC	A
C4AB	3243CA	LD	(CCINV), A
C4AE	FE02	CP	TPINV
C4B0	200A	JR	NZ, SEGUE4
C4B2	AF	XOR	A
C4B3	3243CA	LD	(CCINV), A
C4B6	CDADC5	CALL	INVS

```

C489 CDD8C7      CALL ATBOMB
C48C              SEGUE4:
C48C CD45C8      CALL TESTCH
C48F 2A60CA      LD HL,(FTIRO)
C4C2 3E02        LD A,2
C4C4 8C          CP H
C4C5 2817        JR Z,SEGUE5
C4C7 23          INC HL
C4C8 23          INC HL
C4C9 23          INC HL
C4CA 23          INC HL
C4CB 2260CA      LD (FTIRO),HL
C4CE 5D          LD E,L
C4CF 3E00        LD A,0
C4D1 E5          PUSH HL
C4D2 CD9300      CALL WRTPSG
C4D5 E1          POP HL
C4D6 5C          LD E,H
C4D7 3E01        LD A,1
C4D9 CD9300      CALL WRTPSG
C4DC 1807        JR TSFASE

C4DE              SEGUE5:
C4DE 1E00        LD E,0
C4E0 3E08        LD A,8
C4E2 CD9300      CALL WRTPSG
C4E5              TSFASE:
C4E5 3A38CA      LD A,(CONTD5)
C4E8 FE08        CP LIINV
C4EA 2803        JR Z,TRFASE
C4EC C3D3C3      JP CENTRAL
C4EF              TRFASE:
C4EF CDD8C9      CALL STFASE
C4F2 C3B4C3      JP INFASE

;----- BLOCO 6:ALEATORIZADOR
RANDOM:
C4F5              PUSH HL
C4F5 E5          LD HL,(SEMT)
C4F6 2A52CA      LD DE,(FINSEM)
C4F9 ED5B50CA    AND A
C4FD A7          SBC HL,DE
C4FE ED52        JR NZ,RAND2
C500 2006        LD HL,(INISEM)
C502 2A4ECA      LD (SEMT),HL
C505 2252CA      RAND2:
C508              LD HL,(SEMT)
C508 2A52CA      INC HL
C508 23          LD (SEMT),HL
C50C 2252CA      LD A,(HL)
C50F 7E          POP HL
C510 E1          RET
C511 C9

;----- BLOCO 7:DISPARA TIRO
TRIGGER
C512 CDD800      CALL GTTRIG
C515 A7          AND A
C516 2810        JR Z,NAOPRE
C518 3A36CA      LD A,(DBCUNC)
C51B A7          AND A
C51C 2802        JR Z,LIVRE
C51E AF          XOR A
C51F C9          RET
C520              LIVRE:
C520 3E01        LD A,1
C522 3236CA      LD (DBOUNC),A
C525 3EFF        LD A,255
C527 C9          RET
C528              NAOPRE:
C528 AF          XOR A
C529 3236CA      LD (DBOUNC),A
C52C C9          RET
C52D              TIROS:
C52D 21041B      LD HL,6916
C530 1803        JR TIROS3
C532              TIROS2:
C532 21081B      LD HL,6920
C535              TIROS3:
C535 3A34CA      LD A,(YTR)

```

```

C538 3D          DEC  A
C539 FE00        CP   8
C538 380E        JR   C,FIMTR
C53D 3234CA      LD   (YTR),A
C540 CD4D00      CALL WRTVRM
C543 3A33CA      LD   A,(XTR)
C546 23          INC  HL
C547 CD4D00      CALL WRTVRM
C54A C9          RET
C548             FIMTR:
C54B 3ED1        LD   A,209
C54D 3234CA      LD   (YTR),A
C550 CD4D00      CALL WRTVRM
C553 C9          RET
;----- BLOCO 8:IMPRIME NAVE
IMPNAV:
C554             LD   A,(XNAVE)
C554 3A2DCA      LD   HL,6913
C557 21011B      CALL WRTVRM
C55A CD4D00      LD   HL,6925
C55D 210D1B      CALL WRTVRM
C560 CD4D00      LD   HL,6929
C563 21111B      CALL WRTVRM
C566 CD4D00      LD   A,(YNAVE)
C569 3A2ECA      LD   HL,6912
C56C 21001B      CALL WRTVRM
C56F CD4D00      LD   HL,6924
C572 210C1B      CALL WRTVRM
C575 CD4D00      LD   HL,6928
C578 21101B      ADD  A,14
C57B C60E        CALL WRTVRM
C57D CD4D00      LD   A,(PFOGO),A
C580 3A45CA      INC  A
C583 3C          LD   (PFOGO),A
C584 3245CA      CP   20
C587 FE14        RET  NZ
C589 C0          XOR  A
C58A AF          LD   (PFOGO),A
C58B 3245CA      LD   HL,6930
C58E 21121B      LD   A,(CFOGO)
C591 3A46CA      INC  A
C594 3C          LD   (CFOGO),A
C595 3246CA      AND  1
C598 E601        INC  A
C59A 3C          ADD  A,A
C59B 87          ADD  A,A
C59C 87          CALL WRTVRM
C59D CD4D00      LD   HL,6927
C5A0 210F1B      CALL RDVRM
C5A3 CD4A00      INC  A
C5A6 3C          AND  15
C5A7 E60F        CALL WRTVRM
C5A9 CD4D00      RET
C5AC C9          ;----- BLOCO 9:INVASORES
INVR:
C5AD             XOR  A
C5AD AF          LOOPIN:
C5AE             LD   (LOPINV),A
C5AE 323CCA      LD   HL,ARRAY
C5B1 2168CA      LD   B,0
C5B4 0600        ADD  A,A
C5B6 87          ADD  A,A
C5B7 87          LD   C,A
C5B8 4F          ADD  HL,BC
C5B9 09          LD   (PYINV),HL
C5BA 225ACA      LD   A,(HL)
C5BD 7E          CP   209
C5BE FED1        JR   Z,PROXIN
C5C0 2B65        INC  HL
C5C2 23          LD   (PXINV),HL
C5C3 2258CA      LD   HL,IAPV
C5C6 2188CA      LD   A,(LOPINV)
C5C9 3A3CCA      ADD  A,A
C5CC 87          LD   B,0
C5CD 0600        LD   C,A
C5CF 4F          ADD  HL,BC
C5D0 09

```

C5D1	225CCA	LD	(APV),HL
C5D4	5E	LD	E,(HL)
C5D5	23	INC	HL
C5D6	56	LD	D,(HL)
C5D7	E8	EX	DE,HL
C5D8	7E	LD	A,(HL)
C5D9	FEFF	CP	255
C5D8	2878	JR	Z,FIMINV
C5D0	F5	PUSH	AF
C5DE	3A44CA	LD	A,(FLINC)
C5E1	A7	AND	A
C5E2	2801	JR	Z,NTINC
C5E4	23	INC	HL
C5E5		NTINC:	
C5E5	EB	EX	DE,HL
C5E6	F1	POP	AF
C5E7	323DCA	LD	(DIRI),A
C5EA	2A5CCA	LD	H',(APV)
C5ED	73	LD	(HL),E
C5EE	23	INC	HL
C5EF	72	LD	(HL),D
C5F0	21CCCA	LD	HL,IARX
C5F3	3A3DCA	LD	A,(DIRI)
C5F6	4F	LD	C,A
C5F7	0600	LD	B,0
C5F9	09	ADD	HL,BC
C5FA	46	LD	B,(HL)
C5FB	2A58CA	LD	HL,(PXINV)
C5FE	7E	LD	A,(HL)
C5FF	80	ADD	A,B
C600	77	LD	(HL),A
C601	21D5CA	LD	HL,IARY
C604	3A3DCA	LD	A,(DIRI)
C607	4F	LD	C,A
C608	0600	LD	B,0
C60A	09	ADD	HL,BC
C60B	46	LD	B,(HL)
C60C	2A5ACA	LD	HL,(PYINV)
C60F	7E	LD	A,(HL)
C610	80	ADD	A,B
C611	77	LD	(HL),A
C612	2A58CA	LD	HL,(PXINV)
C615	23	INC	HL
C616	3A3DCA	LD	A,(DIRI)
C619	4F	LD	C,A
C61A	A7	AND	A
C61B	2805	JR	Z,S+7
C61D	C608	ADD	A,B
C61F	87	ADD	A,A
C620	87	ADD	A,A
C621	77	LD	(HL),A
C622	79	LD	A,C
C623	A7	AND	A
C624	C490C7	CALL	NZ,STBOMB
C627		PROXIN:	
C627	3A3CCA	LD	A,(LOPINV)
C62A	3C	INC	A
C62B	FE08	CP	LIINV
C62D	C2AEC5	JP	NZ,LOOPIN
C630	AF	XOR	A
C631	3244CA	LD	(FLINC),A
C634	3A42CA	LD	A,(PSINC)
C637	3C	INC	A
C638	3242CA	LD	(PSINC),A
C63B	FE08	CP	COMPV
C63D	2009	JR	NZ,DSPINV
C63F	3E01	LD	A,1
C641	3244CA	LD	(FLINC),A
C644	AF	XOR	A
C645	3242CA	LD	(PSINC),A
C648		DSPINV:	
C648	2168CA	LD	HL,ARRAY
C64B	11141B	LD	DE,6932
C64E	012000	LD	BC,32
C651	CD5C00	CALL	LDIRVM
C654	C9	RET	

```

C655          FIMINV: LD HL,(IVFASE)
C655 2A56CA    LD DE,32
C658 112000    ADD HL,DE
C658 19        EX DE,HL
C65C EB        LD HL,(APV)
C65D 2A5CCA    LD (HL),E
C660 73        INC HL
C661 23        LD (HL),D
C662 72        LD A,(YIFASE)
C663 3A40CA    LD HL,(PYINV)
C666 2A5ACA    LD (HL),A
C669 77        LD A,(XIFASE)
C66A 3A41CA    INC HL
C66D 23        LD (HL),A
C66E 77        RET
C66F C9

;----- BLOCO 10:MOVE ESTRELAS
C670 ESTRTH: LD HL,PAUSA
C670 218000
C673 TEMPO: DEC HL
C673 20      LD A,H
C674 7C      OR L
C675 85      JR NZ,TEMPO
C676 20FB
C678 ESTRE: LD A,(EVENTO)
C678 3A39CA
C678 LOPEV: DEC A
C678 3D      LD (EVENTO),A
C67C 3239CA  AND A
C67F A7      RET NZ
C680 C0      LD A,10
C681 3E0A    LD (EVENTO),A
C683 3239CA  LD HL,64
C686 214000  CALL ROT
C689 CDB2C6  LD A,(TOGGLE)
C68C 3A3ACA  CPL
C68F 2F      LD (TOGGLE),A
C690 323ACA  AND A
C693 A7      RET Z
C694 C8      LD HL,128
C695 218000  CALL ROT
C698 CDB2C6  LD HL,6918
C69B 21061B  CALL APGFM
C69E CDA4C6  LD HL,6922
C6A1 210A1B
C6A4 APGFM: CALL RDVRM
C6A4 CD4A00  CP 68
C6A7 FE44    RET NZ
C6A9 C0      DEC HL
C6AA 2B      DEC HL
C6AB 2B      LD A,209
C6AC 3ED1    CALL WRTVRM
C6AE CD4D00  RET
C6B1 C9
C6B2 ROT: LD B,63
C6B2 063F    DEC HL
C6B4 2B      CALL RDVRM
C6B5 CD4A00  LD C,A
C6B8 4F
C6B9 LOPROT: DEC HL
C6B9 2B      CALL RDVRM
C6BA CD4A00  INC HL
C6BD 23      CALL WRTVRM
C6BE CD4D00  DEC HL
C6C1 2B      DJNZ LOPROT
C6C2 10F5    LD A,C
C6C4 79      CALL WRTVRM
C6C5 CD4D00  RET
C6C8 C9

;----- BLOCO 11:SABOTADORES
C6C9 SABOT: LD A,(CSAB)
C6C9 3A4BCA  INC A
C6CC 3C      LD (CSAB),A
C6CD 324BCA  CP 150
C6D0 FE96    RET NZ
C6D2 C0

```

C6D3	AF	XOR	A
C6D4	324BCA	LD	(CSAB),A
C6D7	3A4DCA	LD	A,(LISAB)
C6DA	A7	AND	A
C6DB	C8	RET	Z
C6DC	AF	XOR	A
C6DD			
C6DD	324CCA	LD	(NSAB),A
C6E0	0600	LD	B,0
C6E2	4F	LD	C,A
C6E3	21B9CA	LD	HL,ARSAB+1
C6E6	09	ADD	HL,BC
C6E7	09	ADD	HL,BC
C6E8	09	ADD	HL,BC
C6E9	09	ADD	HL,BC
C6EA	7E	LD	A,(HL)
C6EB	FEFF	CP	255
C6ED	2834	JR	Z,PSAB
C6EF	CD5C4	CALL	RANDOM
C6F2	E60F	AND	15
C6F4	284B	JR	Z,ATAQ
C6F6	7E	LD	A,(HL)
C6F7	47	LD	B,A
C6F8	3A2DCA	LD	A,(XNAVE)
C6FB	88	CP	B
C6FC	48	LD	C,B
C6FD	380C	JR	C,SESQ
C6FF	3A4CCA	LD	A,(NSAB)
C702	C602	ADD	A,2
C704	47	LD	B,A
C705	79	LD	A,C
C706	3C	INC	A
C707	10FD	DJNZ	%-1
C709	180A	JR	SSEGUE
C70B			
C70B	3A4CCA	LD	A,(NSAB)
C70E	C602	ADD	A,2
C710	47	LD	B,A
C711	79	LD	A,C
C712	3D	DEC	A
C713	10FD	DJNZ	%-1
C715			
C715	77	LD	(HL),A
C716	23	INC	HL
C717	7E	LD	A,(HL)
C718	FE20	CP	32
C71A	2002	JR	NZ,%+4
C71C	361C	LD	(HL),2B
C71E	46	LD	B,(HL)
C71F	3E34	LD	A,52
C721	90	SUB	B
C722	77	LD	(HL),A
C723			
C723	3A4CCA	LD	A,(NSAB)
C726	3C	INC	A
C727	4F	LD	C,A
C728	3A4DCA	LD	A,(LISAB)
C72B	47	LD	B,A
C72C	79	LD	A,C
C72D	88	CP	B
C72E	38AD	JR	C,LOPSAB
C730	21B8CA	LD	HL,ARSAB
C733	11541B	LD	DE,6996
C736	C920	SLA	B
C738	C820	SLA	B
C73A	48	LD	C,B
C73B	0600	LD	B,0
C73D	CD5C00	CALL	LDIRUM
C740	C9	RET	
C741			
C741	23	INC	HL
C742	3620	LD	(HL),32
C744	DBAA	IN	A,(170)
C746	0632	LD	B,50
C748			
C748	CBBF	RES	7,A


```

C74A D3AA      OUT  (170),A
C74C CBFF      SET  7,A
C74E D3AA      OUT  (170),A
C750 10F6      DJNZ BIP
C752 2B        DEC  HL
C753 46        LD   B,(HL)
C754 3A2DCA     LD   A,(XNAVE)
C757 C608      ADD  A,B
C759 90        SUB  B
C75A FE10      CP   16
C75C 30C5      JR   NC,PSAB
C75E 3E01      LD   A,1
C760 3235CA     LD   (STATX),A
C763 18BE      JR   PSAB
C765 C9        RET

```

;----- BLOCO 12:EXPLOSAO

```

C766          EXPLS:
C766 21021B     LD   HL,6914
C769 3E44      LD   A,68
C76B CD4D00     CALL WRTVRM
C76E 3E0A      LD   A,10
C770 23        TNC  HL
C771 CD4D00     CALL WRTVRM
C774 2128CB     LD   HL,SDEXPL
C777 CD84C9     CALL STSND
C77A 2160EA     LD   HL,60000
C77D          TEXPL:
C77D 2B        DEC  HL
C77E 2B        DEC  HL
C77F 23        INC  HL
C780 7C        LD   A,H
C781 85        OR   L
C782 20F9      JR   NZ,TEXPL
C784 3A49CA     LD   A,(NAVES)
C787 3D        DEC  A
C788 3249CA     LD   (NAVES),A
C78B A7        AND  A
C78C C8        RET  Z
C78D C388C3     JP   INNAVE

```

;----- BLOCO 13:SETA BOMBAS

```

C790          STBOMB:
C790 CDF5C4     CALL RANDOM
C793 E61F      AND  31
C795 A7        AND  A
C796 C0        RET  NZ
C797 CDF5C4     CALL RANDOM
C79A E607      AND  7
C79C 4F        LD   C,A
C79D 3A48CA     LD   A,(LIBOMB)
C7A0 47        LD   B,A
C7A1 79        LD   A,C
C7A2 B8        CP   B
C7A3 D0        RET  NC
C7A4 3247CA     LD   (NBOMB),A
C7A7 B7        ADD  A,A
C7A8 B7        ADD  A,A
C7A9 219BCA     LD   HL,IABOMB
C7AC 0600      LD   B,0
C7AE 4F        LD   C,A
C7AF 09        ADD  HL,BC
C7B0 7E        LD   A,(HL)
C7B1 FED1      CP   209
C7B3 C0        RET  NZ
C7B4 E5        PUSH HL
C7B5 2A5ACA     LD   HL,(PYINV)
C7B8 7E        LD   A,(HL)
C7B9 E1        POP  HL
C7BA 77        LD   (HL),A
C7BB 23        INC  HL
C7BC E5        PUSH HL
C7BD 2A58CA     LD   HL,(PXINV)
C7C0 7E        LD   A,(HL)
C7C1 E1        POP  HL
C7C2 77        LD   (HL),A
C7C3 21CCCA     LD   HL,IARX
C7C6 3A3DCA     LD   A,(DIRI)

```

```

C7C9 4F          LD      C,A
C7CA 09          ADD     HL,BC
C7CB 7E          LD      A,(HL)
C7CC F5          PUSH    AF
C7CD 21C4CA      LD      HL,IADB
C7D0 3A47CA      LD      A,(NBOMB)
C7D3 4F          LD      C,A
C7D4 09          ADD     HL,BC
C7D5 F1          POP     AF
C7D6 77          LD      (HL),A
C7D7 C9          RET

;----- BLOCO 14:MOVE BOMBAS
C7D8             ATBOMB:
C7D8 AF          XOR     A
C7D9             LPBOMB:
C7D9 3247CA      LD      (NBOMB),A
C7DC 87          ADD     A,A
C7DD 87          ADD     A,A
C7DE 0600        LD      B,0
C7E0 4F          LD      C,A
C7E1 2198CA      LD      HL,IABOMB
C7E4 09          ADD     HL,BC
C7E5 7E          LD      A,(HL)
C7E6 FED1        CP      209
C7E8 2B1C        JR      Z,PRBOMB
C7EA 3C          INC     A
C7EB FEAA        CP      170
C7ED 2826        JR      Z,FIMBMB
C7EF 77          LD      (HL),A
C7F0 23          INC     HL
C7F1 E602        AND     2
C7F3 A7          AND     A
C7F4 2010        JR      NZ,PRBOMB
C7F6 E5          PUSH    HL
C7F7 21C4CA      LD      HL,IADB
C7FA 3A47CA      LD      A,(NBOMB)
C7FD 4F          LD      C,A
C7FE 0600        LD      B,0
C800 09          ADD     HL,BC
C801 46          LD      B,(HL)
C802 E1          POP     HL
C803 7E          LD      A,(HL)
C804 80          ADD     A,B
C805 77          LD      (HL),A
C806             PRBOMB:
C806 3A47CA      LD      A,(NBOMB)
C809 3C          INC     A
C80A 4F          LD      C,A
C80B 3A49CA      LD      A,(LIBOMB)
C80E 47          LD      B,A
C80F 79          LD      A,C
C810 8B          CP      B
C811 20C6        JR      NZ,LPBOMB
C813 1804        JR      IMPBMB
C815             FIMBMB:
C815 36D1        LD      (HL),209
C817 1BED        JR      PRBOMB
C819             IMPBMB:
C819 2198CA      LD      HL,IABOMB
C81C 11341B      LD      DE,6964
C81F 012000      LD      BC,32
C822 CD5C00      CALL    LDIRUH
C825 C9          RET

;----- BLOCO 15:INICIA BOMBAS
C826             INIBMB:
C826 AF          XOR     A
C827             LPINIB:
C827 3247CA      LD      (NBOMB),A
C82A 2198CA      LD      HL,IABOMB
C82D 87          ADD     A,A
C82E 87          ADD     A,A
C82F 0600        LD      B,0
C831 4F          LD      C,A
C832 09          ADD     HL,BC
C833 36D1        LD      (HL),209
C835 23          INC     HL

```

```

C836 23          INC  HL
C837 3614        LD   (HL),20
C839 23          INC  HL
C83A 360F        LD   (HL),15
C83C 3A47CA      LD   A,(NBOMB)
C83F 3C          INC  A
C840 FE08        CP   B
C842 20E3        JR   NZ,LPINIB
C844 C9          RET

;----- BLOCO 16:TESTA COLISAO
TESTCH:
C845             CALL RDVDP
C846 CD3E01      BIT   5,A
C848 C86F        RET   Z
C84A C8          LD   A,(YTIRO)
C84B 3A30CA      CP   209
C84E FED1        JR   Z,CPTR2
C850 281D        LD   (YTR),A
C852 3234CA      LD   A,(XTIRO)
C855 3A2FCA      LD   (XTR),A
C858 3233CA      CALL CPTIRO
C85B CDE0C8      LD   (YTIRO),A
C85E 3230CA      LD   A,(FLAC)
C861 3A4ACA      AND  A
C864 A7          JR   Z,CPTR2
C865 2808        LD   HL,691B
C867 21061B      LD   A,68
C86A 3E44        CALL WRTVRM
C86C CD4D00
C86F             CPTR2:
C86F 3A32CA      LD   A,(YTIRO2)
C872 FED1        CP   209
C874 281D        JR   Z,CPCOM
C876 3234CA      LD   (YTR),A
C879 3A31CA      LD   A,(XTIRO2)
C87C 3233CA      LD   (XTR),A
C87F CDE0C8      CALL CPTIRO
C882 3232CA      LD   (YTIRO2),A
C885 3A4ACA      LD   A,(FLAC)
C888 A7          AND  A
C889 2808        JR   Z,CPCOM
C88B 210A1B      LD   HL,6922
C88E 3E44        LD   A,68
C890 CD4D00      CALL WRTVRM
C893             CPCOM:
C893             LD   HL,IABOMB
C896 225ECA      LD   (PTABLE),HL
C899 CDA6C8      CALL CPBMB
C89C 2168CA      LD   HL,ARRAY
C89F 225ECA      LD   (PTABLE),HL
C8A2 CDA6C8      CALL CPBMB
C8A5 C9          RET
C8A6             CPBMB:
C8A6             XOR  A
C8A7             LCBOMB:
C8A7 3247CA      LD   (NBOMB),A
C8AA 87          ADD  A,A
C8AB 87          ADD  A,A
C8AC 0600        LD   B,0
C8AE 4F          LD   C,A
C8AF 2A5ECA      LD   HL,(PTABLE)
C8B2 09          ADD  HL,BC
C8B3 7E          LD   A,(HL)
C8B4 FED1        CP   209
C8B6 281F        JR   Z,CPRBMB
C8B8 47          LD   B,A
C8B9 3A2ECA      LD   A,(YNAVE)
C8BC 90          SUB  B
C8BD C608        ADD  A,B
C8BF FE0A        CP   10
C8C1 3014        JR   NC,CPRBMB
C8C3 23          INC  HL
C8C4 46          LD   B,(HL)
C8C5 3A2DCA      LD   A,(XNAVE)
C8C8 90          SUB  B
C8C9 C608        ADD  A,B
C8CB FE10        CP   16

```

C8CD 3008	JR	NC,CPRBMB
C8CF 2B	DEC	HL
C8D0 36D1	LD	(HL),209
C8D2 3E01	LD	A,1
C8D4 3235CA	LD	(STATX),A
C8D7	CPRBMB:	
C8D7 3A47CA	LD	A,(NBOMB)
C8DA 3C	INC	A
C8DB FE08	CP	8
C8DD 20C8	JR	NZ,LCBOMB
C8DF C9	RET	
C8E0	CPTIRO:	
C8E0 AF	XOR	A
C8E1 324ACA	LD	(FLAC),A
C8E4	LPCPIN:	
C8E4 323CCA	LD	(LOPINV),A
C8E7 87	ADD	A,H
C7E8 87	ADD	A,A
C8E9 0600	LD	B,0
C8EB 4F	LD	C,A
C8EC 2168CA	LD	HL,ARRAY
C8EF 09	ADD	HL,BC
C8F0 7E	LD	A,(HL)
C8F1 FED1	CP	209
C8F3 2822	JR	Z,CPPIN
C8F5 47	LD	B,A
C8F6 3A34CA	LD	A,(YTR)
C8F9 90	SUB	B
C8FA C608	ADD	A,B
C8FC FE10	CP	16
C8FE 3017	JR	NC,CPPIN
C900 23	INC	HL
C901 46	LD	B,(HL)
C902 3A33CA	LD	A,(XTR)
C905 96	SUB	B
C906 C608	ADD	A,B
C908 FE10	CP	16
C90A 3008	JR	NC,CPPIN
C90C 2B	DEC	HL
C90D 36D1	LD	(HL),209
C90F 3E01	LD	A,1
C911 324ACA	LD	(FLAC),A
C914 CD2AC9	CALL	ACERT
C917	CPPIN:	
C917 3A3CCA	LD	A,(LOPINV)
C91A 3C	INC	A
C91B FE08	CP	8
C91D 20C5	JR	NZ,LPCPIN
C91F 3A4ACA	LD	A,(FLAC)
C922 A7	AND	A
C923 3A34CA	LD	A,(YTR)
C926 C8	RET	Z
C927 3ED1	LD	A,209
C929 C9	RET	
C92A	ACERT:	
C92A 3A38CA	LD	A,(CONTD5)
C92D 3C	INC	A
C92E 3238CA	LD	(CONTD5),A
C931 2A62CA	LD	HL,(PONTOS)
C934 ED5B66CA	LD	DE,(PTFASE)
C938 15	ADD	HL,DE
C939 2262CA	LD	(PONTOS),HL
C93C ED5B64CA	LD	DE,(RECORD)
C940 A7	AND	A
C941 ED52	SBC	HL,DE
C943 3804	JR	C,NAOREC
C945 19	ADD	HL,DE
C946 2264CA	LD	(RECORD),HL
C949	NAOREC:	
C949 CD92C9	CALL	PLACAR
C94C C9	RET	
;----- BLOCO 17:INICIA INVASORES		
C94D	INIINU:	
C94D AF	XOR	A
C94E	LOPINI:	
C94E 323CCA	LD	(LOPINV),A

```

C951 87          ADD  A,A
C952 2168CA      LD   HL,ARRAY
C955 0600        LD   B,0
C957 4F          LD   C,A
C958 09          ADD  HL,BC
C959 09          ADD  HL,BC
C95A 3A40CA      LD   A,(YIFASE)
C95D 77          LD   (HL),A
C95E 23          INC  HL
C95F 3A41CA      LD   A,(XIFASE)
C962 77          LD   (HL),A
C963 23          INC  HL
C964 23          INC  HL
C965 3A3FCA      LD   A,(CFASE)
C968 77          LD   (HL),A
C969 2188CA      LD   HL,IAPV
C96C 09          ADD  HL,BC
C96D 225CCA      LD   (APV),HL
C970 2A56CA      LD   HL,(IVFASE)
C973 09          ADD  HL,BC
C974 EB          EX   DE,HL
C975 2A5CCA      LD   HL,(APV)
C978 73          LD   (HL),E
C979 23          INC  HL
C97A 72          LD   (HL),D
C97B 3A3CCA      LD   A,(LOPINV)
C97E 3C          INC  A
C97F FE08        CP   LIINV
C981 20CB        JR   NZ,LOPINI
C983 C9          RET

;----- BLOCO 18:SONS
C984 STSND:      LD   A,(HL)
C984 7E          CP   255
C985 FEFF        CP   Z
C987 CB          RET
C988 23          INC  HL
C989 5E          LD   E,(HL)
C98A E5          PUSH HL
C98B CD9300      CALL WRTPSG
C98E E1          POP  HL
C98F 23          INC  HL
C990 1BF2        JR   STSND

;----- BLOCO 19:PLACARES
C992 PLACAR:     LD   A,8
C992 3E08        OUT  (153),A
C994 D399        LD   A,24
C996 3E18        OUT  (153),A
C998 D399        LD   HL,(PONTOS)
C99A 2A62CA      CALL IMPPL
C99D CDAFC9      LD   A,18
C9A0 3E12        OUT  (153),A
C9A2 D399        LD   A,24
C9A4 3E18        OUT  (153),A
C9A6 D399        LD   HL,(RECORD)
C9A8 2A64CA      CALL IMPPL
C9AB CDAFC9      RET
C9AE C9
C9AF IMPPL:      LD   DE,10000
C9AF 111027      CALL DIGITO
C9B2 CDCAC9      LD   DE,1000
C9B5 11E803      CALL DIGITO
C9B8 CDCAC9      LD   DE,100
C9BB 116400      CALL DIGITO
C9BE CDCAC9      LD   DE,10
C9C1 110A00      CALL DIGITO
C9C4 CDCAC9      LD   DE,1
C9C7 110100      DIGITO: LD   A,48
C9CA 3E30        AND  A
C9CC A7          CONTDI: SBC  HL,DE
C9CD ED52        JR   C,FIMDIG
C9CF 3803        INC  A
C9D1 3C          JR   CONTDI
C9D2 1BF9        FIMDIG:
C9D4

```

```

C9D4 19      ADD HL,DE
C9D5 D398    OUT (152),A
C9D7 C9      RET

```

----- BLOCO 20:INICIA FASE

```

STFASE:
C9D8          LD A,(FASE)
C9D8 3A3ECA   PUSH AF
C9D8 F5       INC A
C9DC 3C       CP 5
C9DD FE05     JR NC,S+5
C9DF 3003     LD (LIBOMB),A
C9E1 3248CA   DEC A
C9E4 3D       SRL A
C9E5 CB3F     SRL A
C9E7 CB3F     SRL A
C9E9 CB3F     AND 3
C9EB E603     JR Z,S+5
C9ED 2803     LD (LISAB),A
C9EF 324DCA   POP AF
C9F2 F1       AND 7
C9F3 E607     LD B,0
C9F5 0600     LD C,A
C9F7 4F       LD HL,TPFASE
C9F8 21FACA   ADD HL,BC
C9FB 09       LD A,(HL)
C9FC 7E       LD (PTFASE),A
C9FD 3266CA   LD HL,IPFASE
CA00 2102CB   ADD HL,BC
CA03 09       ADD HL,BC
CA04 09       LD A,(HL)
CA05 7E       LD (XIFASE),A
CA06 3241CA   INC HL
CA09 23       LD A,(HL)
CA0A 7E       LD (YIFASE),A
CA0B 3240CA   LD HL,ACFASE
CA0E 21F2CA   ADD HL,BC
CA11 09       LD A,(HL)
CA12 7E       LD (CFASE),A
CA13 323FCA   LD HL,DISPLT
CA16 2112CB   ADD HL,BC
CA19 09       ADD HL,BC
CA1A 09       LD E,(HL)
CA1C 23       INC HL
CA1D 56       LD D,(HL)
CA1E 2100A0   LD HL,IPVR
CA21 19       ADD HL,DE
CA22 2256CA   LD (IVFASE),HL
CA25 3A3ECA   LD A,(FASE)
CA28 3C       INC A
CA29 323ECA   LD (FASE),A
CA2C C9      RET

```

----- BLOCO 21:VARIABLES

```

CA2D          VARS:
CA2D 00       XNAVE: DB 0
CA2E 00       YNAVE: DB 0
CA2F 00       XTIRO: DB 0
CA30 00       YTIRO: DB 0
CA31 00       XTIRO2:DB 0
CA32 00       YTIRO2:DB 0
CA33 00       XTR:   DB 0
CA34 00       YTR:   DB 0
CA35 00       STATX: DB 0
CA36 00       DBOUNC:DB 0
CA37 00       MODE:  DB 0
CA38 00       CONTDS:DB 0
CA39 00       EVENTO:DB 0
CA3A 00       TOGGLE:DB 0
CA3B 00       CCAN:  DB 0
CA3C 00       LOPINV:DB 0
CA3D 00       DIRI:  DB 0
CA3E 00       FASE:  DB 0
CA3F 00       CFASE: DB 0
CA40 00       YIFASE:DB 0
CA41 00       XIFASE:DB 0
CA42 00       PSINC: DB 0
CA43 00       CCINV: DB 0

```

CA44	00	FLINC: DB	0
CA45	00	PFOGO: DB	0
CA46	00	CFOGO: DB	0
CA47	00	NBOMB: DB	0
CA48	01	LIBOMB: DB	1
CA49	00	NAVES: DB	0
CA4A	00	FLAC: DB	0
CA4B	00	CSAB: DB	0
CA4C	00	NSAB: DB	0
CA4D	01	LISAB: DB	1
;			
CA4E	0040	INISEM: DW	16384
CA50	FF7F	FIMSEM: DW	32767
CA52	0000	SEMT: DW	0
CA54	0000	CVRAM: DW	0
CA56	0000	IVFASE: DW	0
CA58	0000	PXINU: DW	0
CA5A	0000	PYINU: DW	0
CA5C	0000	APV: DW	0
CA5E	0000	PTABLE: DW	0
CA60	0000	FTIRO: DW	0
CA62	0000	PONTOS: DW	0
CA64	0000	RECORD: DW	0
CA66	0000	PTFASE: DW	0
;----- BLOCO 22: ARRAYS			
CA6B		ARRAY: DS	32
CA8B		IAPV: DS	16
CA9B		IABOMB: DS	32
CA8B	AF141E02	ARSAB: DB	175,20,24,2,175,80,28,13,175,22
CABC	AF501C0D		
CAC0	AFDC180B		
CAC4		IADB: DS	8
CACC	00000101	IARX: DB	0,0,1,1,1,0,-1,-1,-1
CAD0	0100FFFF		
CAD4	FF		
CAD5	00FFFFFF00	IARY: DB	0,-1,-1,0,1,1,1,0,-1
CAD9	01010100		
CADD	FF		
CADE	A0B0000E	ARPC: DB	160,128,0,14,209,128,16,15,209, 128,16,15,160,128,12,0,176,128,16,8
CAE2	D180100F		
CAE6	D180100F		
CAEA	A0B00C00		
CAEE	B0B0100B		
CAF2	0B04020A	ACFASE: DB	8,4,2,10,6,5,3,15
CAF6	0605050F		
CAFA	323C4650	TPFASE: DB	50,60,70,80,90,100,110,120
CAFE	5A646E7B		
CB02	FF465AF0	IPFASE: DB	255,70,90,240,255,20,90,207,90, 240,255,20,120,207,110,240
CB06	FF1451CF		
CB0A	5AF0FF14		
CB0E	78CF6EF0		
CB12	00009B00	DISPLT: DW	0,9BH,14CH,200H,2BEH,370H,453H,
CB16	4C010002		
CB1A	8E027003		
CB1E	53040E05		
CB22	07FE0000	SDTIRO: DB	7,254,0,0,1,0,8,15,255
CB26	01000D0F		
CB2A	FF		
CB2B	07F7061F	SDEXPL: DB	7,247,6,31,8,16,12,32,13,0,255
CB2F	0B100C20		
CB33	0U00FF		
;			
CB36	2020504F	HSTEL: DB	' PONTOS: 00000 '
CB3A	4E544F53		
CB3E	3A303030		
CB42	303020		
CB45	4D415B3A	DB	' MAX: 00000 NAVES: '
CB49	30303030		
CB4D	30204E41		
CB51	5645533A		
CB55	20		
CB56		END	

E, finalmente, a figura 6.5. Volte novamente ao monitor e use M &HA000. Após digitar, comande

BSAVE "TRAJ",&HA000;&HA5A1

FIGURA 6.5 - Trajetória dos invasores

```

A000 00 00 00 00 00 00 00 00
A008 00 00 00 00 00 00 00 00
A010 00 00 00 00 00 00 00 00
A018 00 00 00 00 00 00 00 00
A020 00 02 02 02 02 02 03 03
A028 04 04 05 05 05 05 05 06
A030 06 05 05 05 04 04 03 04
A038 04 03 02 02 01 01 01 01
A040 01 01 01 01 00 00 07 06
A048 06 05 05 05 05 05 04 04
A050 03 02 02 02 02 01 01 01
A058 01 01 01 01 01 01 08 08
A060 07 06 06 05 05 05 05 05
A068 05 05 04 04 03 02 02 02
A070 02 03 04 05 05 05 05 05
A078 05 05 05 06 06 07 08 08
A080 01 01 01 01 01 01 01 01
A088 02 02 02 03 04 04 04 04
A090 03 02 02 02 02 02 02 FF
A098 00 00 00 00 00 00 00 00
A0A0 00 00 00 00 00 00 00 00
A0AB 00 00 00 00 00 00 00 00
A0B0 00 00 00 00 00 00 00 00
A0B8 05 05 06 06 06 06 06 06
A0C0 06 05 05 04 04 04 04 03
A0C8 03 02 02 02 02 03 03 04
A0D0 04 04 04 05 06 06 06 06
A0D8 07 07 08 08 08 08 07 07
A0E0 07 08 08 08 08 08 01 01
A0E8 02 02 03 03 03 03 04 04
A0F0 04 04 04 04 04 04 04 04
A0F8 04 05 06 06 06 06 07 07
A100 07 08 08 08 08 08 08 08
A108 01 01 01 01 02 02 02 02
A110 02 02 02 03 03 04 04 04
A118 04 04 04 04 04 05 05 05
A120 06 06 06 06 06 06 07 07
A128 07 08 08 07 06 06 05 05
A130 04 03 03 03 02 02 02 02
A138 02 01 01 01 01 01 01 08
A140 08 08 08 08 08 08 08 08
A148 08 08 08 FF 00 00 00 00
A150 00 00 00 00 00 00 00 00
A158 00 00 00 00 00 00 00 00
A160 00 00 00 00 00 00 00 00
A168 00 00 00 00 03 03 03 03
A170 03 03 04 05 04 03 03 03
A178 03 03 03 04 05 04 03 03
A180 03 03 03 04 05 04 03 03
A188 03 03 03 03 04 05 06 07
A190 07 07 07 07 07 08 01 08
A198 07 07 07 07 07 07 08 01
A1A0 08 07 07 07 07 07 07 06
A1AB 05 04 03 03 03 03 03 03
A1B0 04 05 04 03 03 03 03 03
A1B8 03 04 05 04 03 03 03 03
A1C0 03 03 04 05 06 07 07 07
A1C8 07 07 07 08 01 08 07 07
A1D0 07 07 07 07 06 05 06 07
A1D8 07 07 07 07 07 08 01 01
A1E0 01 01 01 01 01 02 02 02
A1E8 02 02 02 02 02 03 03 03
A1F0 04 03 03 03 03 03 04 03
A1F8 03 03 03 03 02 02 02 FF
A200 00 00 00 00 00 00 00 00
A208 00 00 00 00 00 00 00 00
A210 00 00 00 00 00 00 00 00

```


A218	00	00	00	00	00	00	00	00
A220	01	01	01	01	01	01	01	01
A228	01	01	01	01	01	01	01	01
A230	01	01	01	01	01	01	01	08
A238	06	05	04	03	03	02	01	08
A240	07	06	05	04	03	03	04	05
A248	05	05	05	06	06	06	06	06
A250	06	06	05	04	04	03	03	03
A258	03	03	02	02	02	01	01	01
A260	01	01	01	01	02	02	02	03
A268	03	03	04	04	04	05	05	05
A270	06	06	06	07	07	07	08	08
A278	01	01	02	02	03	03	04	05
A280	06	07	08	01	00	00	00	00
A288	00	00	00	00	05	05	05	05
A290	05	05	05	05	05	02	01	01
A298	01	01	01	01	01	01	01	01
A2A0	02	02	01	01	01	08	07	06
A2A8	05	04	03	04	05	05	05	05
A2B0	05	05	05	04	04	05	05	05
A2B8	05	05	05	05	05	FF	00	00
A2C0	00	00	00	00	00	00	00	00
A2C8	00	00	00	00	00	00	00	00
A2D0	00	00	00	00	00	00	00	00
A2D8	00	00	00	05	05	05	05	05
A2E0	05	05	05	05	05	08	08	07
A2E8	07	06	06	05	05	04	04	03
A2F0	03	02	08	06	06	05	05	04
A2F8	04	03	03	02	02	01	01	08
A300	08	06	04	04	03	03	02	02
A308	01	01	08	08	07	07	06	06
A310	05	05	04	04	04	04	03	03
A318	03	02	02	01	01	08	08	07
A320	07	06	06	05	05	05	05	05
A328	04	04	03	03	00	00	00	00
A330	03	03	02	02	01	01	02	02
A338	02	03	03	04	04	05	05	05
A340	06	06	05	05	06	06	06	07
A348	00	00	00	00	00	00	00	00
A350	00	00	00	00	00	00	00	00
A358	08	01	01	01	01	01	01	01
A360	01	01	01	01	01	01	01	08
A368	01	01	01	01	01	01	01	FF
A370	00	00	00	00	00	00	00	00
A378	00	00	00	00	00	00	00	00
A380	00	00	00	00	00	00	00	00
A388	00	00	00	00	00	00	00	00
A390	03	03	03	03	04	04	03	03
A398	03	03	02	02	02	03	03	03
A3A0	03	03	04	04	04	05	05	05
A3A8	05	05	06	06	06	07	07	07
A3B0	07	07	08	08	08	01	01	01
A3B8	01	01	02	02	02	03	03	03
A3C0	03	03	04	04	04	05	05	05
A3C8	05	05	06	06	06	07	07	07
A3D0	07	07	08	08	08	01	01	01
A3D8	01	01	01	01	02	02	02	03
A3E0	03	04	05	05	05	05	05	05
A3E8	05	05	04	03	03	02	01	01
A3F0	08	07	07	06	05	05	04	03
A3F8	03	02	01	01	08	07	07	06
A400	05	05	04	03	03	03	03	02
A408	03	04	05	05	05	05	05	06
A410	07	07	06	05	05	05	06	07
A418	07	08	01	01	01	08	07	07
A420	06	05	05	05	06	07	07	08
A428	01	01	01	08	07	07	06	05
A430	05	05	05	00	00	00	00	00
A438	00	00	00	01	01	01	01	01
A440	01	01	01	01	01	01	01	01
A448	02	01	01	01	01	01	08	07
A450	07	07	FF	00	00	00	00	00
A458	00	00	00	00	00	00	00	00
A460	00	00	00	00	00	00	00	00
A468	00	00	00	00	00	00	00	00
A470	00	00	00	01	01	01	01	01

```

A478 01 01 01 01 01 01 01 01
A480 01 03 03 03 03 07 07 07
A488 07 07 07 07 07 03 03 03
A490 03 05 05 05 05 05 05 07
A498 06 05 05 05 06 07 07 07
A4A0 08 01 01 01 08 07 06 05
A4A8 05 05 04 03 03 07 07 07
A4B0 07 07 08 01 01 01 01 01
A4B8 02 02 02 02 02 02 03 03
A4C0 03 03 02 03 03 03 03 03
A4C8 03 04 03 03 04 05 05 05
A4D0 05 05 05 05 01 01 01 08
A4D8 07 07 06 05 05 05 05 05
A4E0 05 04 03 03 03 03 02 01
A4E8 01 01 01 01 01 01 01 01
A4F0 01 01 01 08 08 08 01 01
A4F8 08 07 07 07 06 05 05 06
A500 07 07 08 01 01 02 03 02
A508 01 08 01 01 01 FF 00 00
A510 00 00 00 00 00 00 00 00
A518 00 00 00 00 00 00 00 00
A520 00 00 00 00 00 00 00 00
A528 00 00 00 00 05 05 05 06
A530 05 05 05 06 05 05 05 06
A538 05 05 05 06 05 05 05 06
A540 05 04 03 03 03 03 02 03
A548 03 02 03 03 02 03 03 02
A550 03 03 02 03 03 04 05 00
A558 00 00 00 00 00 00 00 00
A560 00 00 00 00 00 00 05 05
A568 05 05 05 06 07 07 07 07
A570 07 07 07 07 07 07 07 07
A578 07 07 07 07 07 07 07 07
A580 07 07 07 07 07 07 07 08
A588 01 01 01 01 01 01 01 01
A590 01 01 01 01 01 01 01 01
A598 01 01 01 01 01 01 01 01
A5A0 01 FF

```

Pronto, agora é só rodar. Antes disso, certifique-se de que os programas foram salvos corretamente. Digite então: RUN "ORPHEUS" (para disco)

Se você quiser gravar os programas em fita cassete, use "CAS:" como nome de dispositivo em todas as gravações e altere a linha 200 do programa listado na figura 6.2 para:

```

200 BLOAD"CAS:TELA":BLOAD"CAS:JOGO":
BLOAD"CAS:TRAJ"

```

Se tudo estiver correto o jogo começará após alguns segundos. Seus controles são as teclas de cursor e a barra de espaços, mas podem ser mudados para joystick com o comando

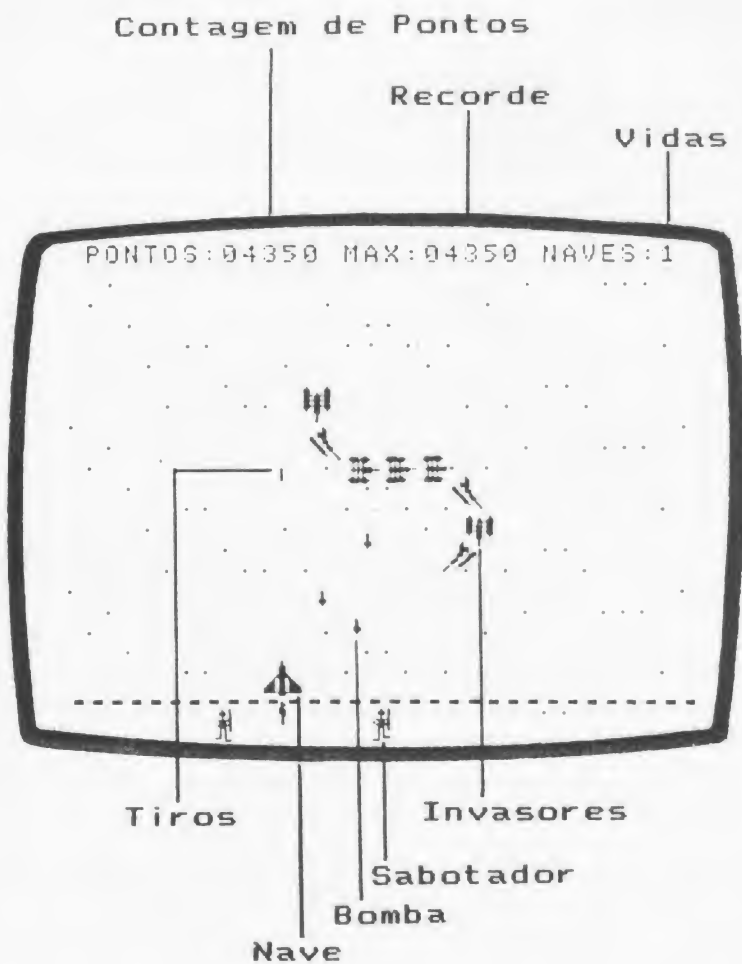
```
POKE &HCA4C,1
```

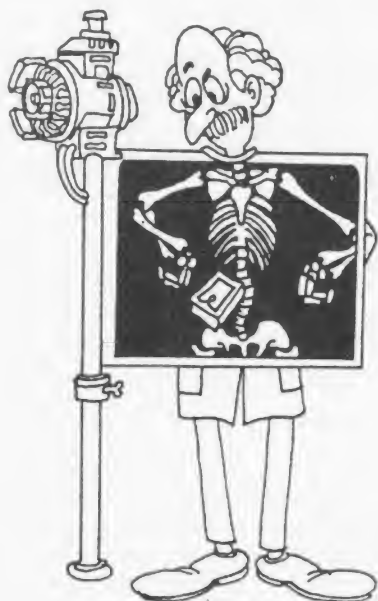
e voltados para o teclado com

```
POKE &HCA4C,0
```

Defenda a Terra com toda a sua garra e que a Força esteja com você, sempre!

FIGURA 6.6 - "Lay Out" da tela do ORPHEUS





APÊNDICES

APÊNDICE I	- MAPA DAS PORTAS DO Z80	138
APÊNDICE II	- VARIÁVEIS DO SISTEMA	141
APÊNDICE III	- MAPA DOS HOOKS DO MSX	147
APÊNDICE IV	- MONITOR ASSEMBLER	150

APÊNDICE I - MAPA DAS PORTAS DO Z80

O processador Z80 é capaz de manipular 256 portas de entrada e saída diferentes. Quando se usa o comando OUT N,A do processador, a porta N recebe o dado fornecido em A. Reciprocamente, o comando IN A,(N) lê o dado da porta N e o coloca no acumulador. N pode estar entre 0 e 255.

Para operar os circuitos periféricos do computador, o Z80 possui circuitos de controle ligados às suas portas. Assim, por exemplo, o teclado usa as portas 169 e 170 (&HA9 e &HAA, respectivamente).

O mapa completo das portas de controle é o seguinte:

&HFF	Vazio
&HF8	* Controle de áudio e vídeo
&HF0	Vazio
&HE0	* Rom de caracteres japoneses
&HD8	Vazio
&HC0	Interface de light-pen
&HB8	* Memórias externas
&HB0	PPI (P8255A)
&HA8	PSG (AY-3-8910)
&HA0	VDP (TMS9128)
&H98	Interface de impressora
&H90	Vazio
&H88	Interface RS-232C
&H80	Não utilizado
&H00	

I. As portas assinaladas com um "*" são usadas apenas por algumas marcas japonesas. Nos micros nacionais (Expert e Hot-Bit) elas estão desligadas.

II. As portas situadas entre &H00 e &H7F não são usadas pelo sistema MSX.

Descrição das portas do MSX:

1. RS-232C

80H Porta de dados do 8251.
 81H Porta de comandos/status da 8251.
 82H Chaves de seleção de velocidade.
 bits 0-3: velocidade de
 transmissão.
 bits 4-7: velocidade de
 recepção.

valor	velocidade (bauds)	valor	velocidade (bauds)
0	50	8	4800
1	75	9	9600
2	110	A	12800
3	150	B	-
4	300	C	-
5	600	D	-
6	1200	E	-
7	2400	F	desligado

83H Chaves de seleção da configuração
 (modo leitura).
 Registro de interrupção (modo
 escrita).
 Modo leitura - funções diversas.

Bit	Função	Lógica
0	CD (detetor de sinal).	
1	ativa/desat. auto line feed na recepção.	1=Ativa.
2	Full/Half duplex.	1=Full duplex.
3	Liga/desliga controle externo.	1=Liga.
4	Tamanho da palavra.	1=8 bits; 0=7 bits.
5	Paridade par/ímpar.	1=Par.
6	Liga/desliga paridade.	1=Liga.
7	Stop bits.	1=2 bits; 0=1 bit.

84H Contador 0 do 8253.
 85H Contador 1 do 8253.
 86H Contador 2 do 8253.
 87H Registro de modo do
 8253.

2. IMPRESSORA

90H	Linha "BUSY": bit 1 (leitura).
90H	Saída "STROBE": bit 0 (escrita).
91H	Saída de dados.

3. VDP

98H	Dados da VRAM.
99H	Endereços e status da VRAM e VDP.

4. PSG

A0H	Endereço (número do registro).
A1H	Saída de dados.
A2H	Entrada de dados.

5. PPI

A8H	Porta A.
A9H	Porta B.
AAH	Porta C.
ABH	Porta de controle.

APÊNDICE II - VARIÁVEIS DO SISTEMA

Este apêndice contém o mapa das variáveis do sistema mais importantes para o usuário.

A notação da área de variáveis é a seguinte: à esquerda há um número em hexadecimal que aponta o primeiro endereço e em seguida há o nome. O número que segue o nome indica o tamanho da variável em bytes (em decimal).

Inicialmente temos as sub-rotinas que permitem, de maneira simples, ler, escrever e chamar qualquer endereço de qualquer SLOT primário. Essas sub-rotinas são colocadas na RAM durante a inicialização do sistema e podem ser utilizadas pelo usuário.

PPI.AW=&B10101000=&HAB ; Porta A da PPI.

```
F380 RDPRM, 5
      OUT (PPI.AW), A ; Escolhe SLOT primário.
      LD E, (HL)      ; Lê endereço dado por HL.
      JR WRPRM1       ; Volta ao estado inicial.
F385 WRPRM, 7
      OUT (PPI.AW), A ; Escolhe SLOT primário.
      LD (HL), E      ; Escreve o dado.
      WRPRM1
      LD A, D          ; Carrega o estado inicial.
      OUT (PPI.AW), A ; Volta ao estado inicial.
      RET
F38C CLPRIM, 14
      OUT (PPI.AW), A
      EX AF, AF'      ; Salva acumulador e flags.
      CALL CLPRIM+12 ; exec. CALL através de IX.
      EX AF, AF'      ; Salva regs remanescentes.
      POP AF          ; Recupera estado inicial.
      OUT (PPI.AW), A
      EX AF, AF'      ; Volta regs remanescentes.
      RET
      JP (IX)         ; Chamado pelo CALL acima.
```

Seguem as variáveis do sistema.

```
F39A USRTAB, 20
      DW FCERR
      CW FCERR
      DW FCERR
      DW FCERR
      DW FCERR
      DW FCERR
      DW FCERR
```


DW FCERR
 DW FCERR
 DW FCERR

F3AE	LINL40,1	DB 39	;Largura da SCREEN 0.
F3AF	LINL32,1	DB 29	;Largura da SCREEN 1.
F3B0	LINLEN,1	DB 39	;Largura corrente.
F3B1	CRTCNT,1	DB 24	;Número de linhas da tela.
F3B2	GLMLST,1	DB 14	

Dados da SCREEN 0.

F3B3	TXTNAM,2	DB 0	;Endereço do mapa da tela.
F3B5	TXTCOL,2	DW 0	;Sem uso.
F3B7	TXTCGP,2	DW 2048	;End. tabela de caracteres.
F3B9	TXATTR,2	DW 0	;Sem uso.
F3BB	TXTPAT,2	DW 0	;Sem uso.

Dados da SCREEN 1.

F3BD	T32NAM,2	DW 6144	;End. do mapa da tela.
F3BF	T32COL,2	DW 8192	;End. do mapa de cores.
F3C1	T32CGP,2	DW 0	;End. tabela de caracteres.
F3C3	T32ATR,2	DW 6912	;End. de atributos sprites.
F3C5	T32PAT,2	DW 14336	;End. tab. padrões sprites.

Dados da SCREEN 2.

F3C7	GRPNAM,2	DW 6144	;Endereço do mapa da tela.
F3C9	GRPCOL,2	DW 8192	;Endereço do mapa de cores.
F3CB	GRPCGP,2	DW 0	;End. tabela de caracteres.
F3CD	GRPATR,2	DW 6912	;End. de atributos sprites.
F3CF	GRPPAT,2	DW 14336	;End. tab. padrões sprites.

Dados da SCREEN 3.

F3D1	MLTNAM,2	DW 2048	;Endereço do mapa da tela.
F3D3	MLTCOL,2	DW 0	;Sem uso.
F3D5	MLTCGP,2	DW 0	;End. tabela de caracteres.
F3D7	MLTATR,2	DW 6912	;End. atributos sprites.
F3D9	MLTPAT	DW 14336	;End. tab. padrões sprites.
F3DB	CLIKSW,1	DB 1	;Chaveia o som do teclado.
F3DC	CSRY,1	DB 1	;Posição Y do cursor.
F3DD	CSRX,1	DB 1	;Posição X do cursor.
F3DE	CNSDFG,1	DB 255	;Indica se deve imprimir as teclas de função.

Cópia dos registros do VDP.

F3DF	RG0SAV,1	DB 0	
F3E0	RG1SAV,1	DB 0	
F3E1	RG2SAV,1	DB 0	
F3E2	RG3SAV,1	DB 0	
F3E3	RG4SAV,1	DB 0	
F3E4	RG5SAV,1	DB 0	
F3E5	RG6SAV,1	DB 0	
F3E6	RG7SAV,1	DB 0	
F3E7	STATFL,1	DB 0	
F3E8	TRGFLG,1	DB 241	;Indica pressão na barra de ; espaço/botão do joystick.
F3E9	FORCLR,1	DB 15	;Cor de frente da tela.
F3EA	BAKCLR,1	DB 4	;Cor de fundo da tela. ;(DB 1 para o Expert)
F3EB	BDRCLR,1	DB 4	;Cor da borda da tela. ;(DB 1 para o Expert)
F3F6	SCNCNT,1	DB 1	;Intervalo leitura teclas.
F3F7	REPCNT,1	DB 50	;Intervalo p/ auto-repeat.
F3F8	PUTPNT,2	DW KEYBUF	;Endereço do último caract- ; ter digitado.
F3FA	GETPNT,2	DW KEYBUF	;Endereço do próximo caract- ; ter lido pelo sistema.
F3FC	CS120,10		;Parâmetros do cassete.

Constantes para 1200 BPS.

LOW01= 83;Tempo pulso baixo bit 0.
 HIGHT01=92;Tempo pulso alto bit 0.
 LOW11= 38;Tempo pulso baixo bit 1.
 HIGHT11=45;Tempo pulso alto bit 1.

Constantes para 2400 BPS.

LOW02= 37;Tempo pulso baixo bit 0.
 HIGH02= 45;Tempo pulso alto bit 0.
 LOW12= 14;Tempo pulso baixo bit 1.
 HIGH12= 22;Tempo pulso alto bit 1.

HEDLEN=2000;Tam.(em bits) do header.

DB LOW01
 DB HIGH01
 DB LOW11
 DB HIGH11
 DB HEDLEN*2/256

```

                                DB LOW02
                                DB HIGH02
                                DB LOW12
                                DB HIGH12
                                DB HEDLEN*4/256
F406  LOW,2
                                DB LOW01
                                DB HIGH01
F408  HIGH,2
                                DB LOW11
                                DB HIGH11
F40A  HEADER,1
                                DB HEDLEN*2/256

```

ENDPRG :última posição que exige inicialização.

```

F40F  ENDPRG,5    DW ":"      ;Assume fim do programa
                                ;para RESUME NEXT.

```

Final das constantes inicializadas.

```

F414  ERRFLG,1    ;Salva o número do erro.
F415  LPTPOS,1    ;Coluna da impressora
                                ;(iniciada com 0).
F416  PRTFLG,1    ;Indica saída p/impressora.
F417  NTMSXP,1    ;Vale 0 p/ impressora MSX.
                                ;No Expert 1.1 e no HOTBIT
                                ;o funcionamento dessa
                                ;variável de sistema é o
                                ;inverso. Quando em 0 estão
                                ;ativos os filtros para a
                                ;compatibilização ABNT ou
                                ;ABICOMP.
F418  RAWPRT,1    ;Não-0 durante "raw mode".
F419  VLZADR,2    ;End. carac. usado por VAL.
F41B  VLZDAT,1    ;Carac. trocado por 0
                                ;por VAL.
F672  MEMSIZ,2    ;Maior posição da memória.
F674  STKTOP,2    ;Maior posição designada p/
                                ;o "stack", inicializada de
                                ;acordo com MEMSIZ.
F676  TXTTAB,2    ;Início do programa BASIC.
F6A3  DATLIN,2    ;No. da linha com erro.
F6B9  ONELIN,2    ;A linha do "ON ERROR
                                ;GOTO".
F6BB  ONEFLG,1    ;Assume 1 quando há
                                ;ONERROR.
F6C2  VARTAB,2    ;Aponta área de variáveis
                                ;simples. Recebe (TXTTAB)+2
                                ;ao se iniciar o BASIC

```

F6C4	ARYTAB,2	; (ou NEW) e muda com o ; tamanho do programa. ; Aponta tabela de "arrays". ; Recebe (VARTAB) ao se ; limpar a área de ; variáveis.
F6C6	STREND,2	; Marca o final da área ; ativa das variáveis.
F6CA	DEFTBL,26	; Indica o tipo da variável ; que começa com a respec- ; tiva letra.
F7C4	TRCFLG,1	; Marca 1 quando TRON ativo.
F85F	MAXFIL,1	; Número máximo de arquivos.
F860	FILTAB,2	; End. dos dados do arquivo.
F8C2	NULBUF,2	; End. do buffer do canal 0.
F864	PTRFIL,2	; End. dos dados do arquivo ; chamado.
F866	RUNFLG,0	; Indica se o programa deve ; rodar após a leitura.
F866	FILNAM,11	; Usado pelo comando NAME, ; guarda o nome do arquivo ; a alterar.
F871	FILNM2,11	; Guarda o novo nome do ; arquivo.
F87C	NLONLY,1	; Não nulo quando lendo ; programa.
F87D	SAVEND,2	; Último byte do BSAVE.
F87F	FNKSTR,160	; Área das teclas de função.
F91F	CGPNT,3	; Endereço e slot da tabela ; de caracteres (usada ; pelo SCREEN).
F922	NAMBAS,2	; Base do mapa de tela.
F924	CGPBAS,2	; Base da tab. caracteres.
F926	PATBAS,2	; Base da tab. padrões ; sprites.
F928	ATRBAS,2	; Base da tab. atrib. ; sprites.
FBB0	ENSTOP,1	; Flag de habilit. do ; warmstart. Quando ; diferente de zero, e as ; teclas CTRL+SHIFT+RGRA(ou ; CODE)+LGRA(ou GRAPH) são ; pressionadas ; simultaneamente o controle ; do micro é devolvido ao ; usuário, desde que a ; interrupção ; esteja habilitada.
FBB1	BASROM,1	; Não zero para cartuchos ; BASIC

FB82	LINTTB,24	; Comprimento das linhas ; lógicas da tela.
FBCA	FSTPOS,2	; Primeira posição para ; INLIN.
FBCC	CODSAV,2	; Cód. do caracter sob o ; cursor.
FBCD	FNKSWI,1	; Indica quando imprimir as ; teclas de função.
FBCE	FNKFLG,1	; Indica as teclas ativas ; por KEY (N) ON.
FBD8	ONGSBF,1	; Indica se ON KEY GOSUB ; está ativo.
FBD9	CLIKFL,1	; Indica "click" de teclado.
FC48	BOTTOM,2	; Indica início da RAM.
FC4A	HIMEM,2	; Indica final da RAM ativa.
FCA8	INSFLG,1	; Indica modo de inserção.
FGA9	CSRSW,1	; Indica 1 ao acender ; cursor.
FCAA	CSTYLE,1	; Indica o tipo do cursor.
FCAB	CAPST,1	; Trava/destrava maiúsculas.
FCAF	SCRMOD,1	; Guarda o número da SCREEN.
FCBF	SAVENT,2	; End. inicial do BSAVE.
FCC1	EXPTBL,4	; Tab. flags slots ; expandidos.
FCC5	SLTTBL,4	; Tab. dos slots secundários ; atuais de cada primário.
FCC9	SLTATR,64	; Guarda atributos das ; páginas dos slots.
FD09	SLTWRK,128	; Area de trabalho para ; os slots.
FD89	PROCNAM,16	; Armazena o nome digitado ; após o comando CALL.
FD99	DEVICE,1	; Identificador de perifé- ; rico para cartucho (0-3).

A seguir inicia-se a área dos hooks.

APÊNDICE III - MAPA DOS HOOKS DO MSX

Os hooks do sistema MSX estão listados a seguir, juntamente com seus endereços de chamada e uma breve explicação de suas funções. Cada um dos hooks possui 5 bytes. No capítulo 1 há uma explicação detalhada da utilização dos hooks.

FD9A	HKEYI	00C4	Vetor de interrupção 60 Hz.
FD9F	HTIMI	0C53	Vetor de interrupção 60 Hz.
FDA4	HGHPU	08C0	CHPUT do BIOS.
FDA9	HDSPC	09E6	Imprime de cursor.
FDAE	HERAC	0A33	Apaga cursor.
FDB3	HDSPF	0B2B	DSPFNK do BIOS.
FDB8	HERAF	0B15	ERAFNK do BIOS.
FDBD	HTOTE	0B42	TOTEXT do BIOS.
FDC2	HCHGE	10CE	CHGET do BIOS.
FDC7	HINIP	071E	Copia tab. caracteres p/ VRAM.
FDCC	HKEYC	1025	Leitor de teclado.
FDD1	HKEYEA	0F10	Leitor de teclado.
FDD6	HNMI	1398	NMI do BIOS.
FDD8	HPINL	23BF	PINLIN do BIOS.
FDE0	HQINL	23CC	QINLIN do BIOS.
FDE5	HINLI	23D5	INLIN do BIOS.
FDEA	HONGO	7810	"ON DEVICE GOSUB"
FDEF	HDSKO	7C16	"DSKO\$"
FDF4	HSETS	7C1B	"SET"
FDF9	HNAME	7C20	"NAME"
FDEF	HKILL	7C25	"KILL"
FE03	HIPL	7C2A	"IPL"
FE08	HCOPY	7C2F	"COPY"
FE0D	HGMD	7C34	"CMD"
FE12	HDSKF	7C39	"DSKF"
FE17	HDSKI	7C3E	"DSKI\$"
FE1C	HATTR	7C43	"ATTR\$"
FE21	HLSET	7C48	"LSET"
FE26	HRSET	7C4D	"RSET"
FE2B	HFIELD	7C52	"FIELD"
FE30	HMKI\$	7C57	"MKI\$"
FE35	HMK\$	7C5C	"MKS\$"
FE3A	HMKD\$	7C61	"MKD\$"
FE3F	HCVI	7C66	"CVI"
FE44	HCVS	7C6B	"CVS"
FE49	HCVD	7C70	"CVD"
FE4E	HGETP	6A93	Localizador do FCB.
FE53	HSETF	6AB3	Localizador do FCB.
FE58	HNOFO	6AF6	"OPEN"

FE5D	HNULO	6BF0	"OPEN"
FE62	HNTFL	6B3B	Fecha buffer 0 de E/S.
FE67	HMERG	6B63	"MERGE/LOAD"
FE6C	HSAVE	6BA6	"SAVE"
FE71	HBINS	6BCE	"SAVE"
FE76	HBINL	6BD4	"MERGE/LOAD"
FE7B	HFILE	6C2F	"FILES"
FE80	HDGET	6C3B	"GET/PUT"
FE85	HFILO	6C51	Gravação sequencial.
FE8A	HINDS	6C79	Leitura sequencial.
FE8F	HRSLF	6CD8	"INPUT\$"
FE94	HSAVD	6D03	"LOC"
		6D14	"LOF"
		6D25	"EOF"
		6D39	"FPOS"
FE99	HLOC	6D0F	"LOC"
FE9E	HLOF	6D20	"LOF"
FEA3	HEOF	6D33	"EOF"
FEA8	HFPOS	6D43	"FPOS"
FEAD	HBAKU	6E36	"LINE INPUT #"
FEB2	HPARD	6F15	Analizador do nome do periférico.
			Idem.
FEB7	HNODE	6F33	Idem.
FEBC	HPOSD	6F37	Sem uso.
FEC1	HDEVN		Funções de E/S.
FEC6	HGEND	6F8F	RUN-CLEAR.
FECB	HRUNC	629A	Idem.
FED0	HGLEA	62A1	Idem.
FED5	HLOPD	62AF	Idem.
FEDA	HSTKE	62F0	Limpa stack.
FEDF	HISFL	145F	ISFLIO do BIOS.
FEE4	HOUTD	1B46	OUTDO do BIOS.
FEE9	HCRDO	732B	OUTDO (com CR ou LF).
EEEE	HDSKC	7374	Loop de entrada de linha.
			Traçado de linhas.
FEF3	HDOGR	593C	Fim do programa.
FEF8	HPRGE	4039	Operador de erros.
FEFD	HERRP	40DC	Idem.
FF02	HERRF	40FD	Controle de "prompt"
FF07	HREAD	412B	("Ok").
			Loop principal.
FF0C	HMAIN	4134	Controle de comando
FF11	HDIRD	41A8	direto.
			Final do loop
FF16	HFINI	4237	principal.
			Idem.
FF1B	HFINE	4247	Varredor de "tokens"
FF20	HCRUN	42B9	Idem.
FF25	HCRUS	4353	Idem.
FF2A	HISRE	437C	Idem.

FF2F	HNTFN	43A4	Idem.
FF34	HNOTR	44EB	Idem.
FF39	HSNGF	45D1	"FOR".
FF3E	HNEWS	4601	Nova instrução a ser executada.
FF43	HGONE	4646	Execução do programa BASIC.
FF48	HCHRG	4666	CHRGTR do BIOS.
FF4D	HRETU	4821	"RETURN"
FF52	HPRTF	4A5E	"PRINT"
FF57	HCOMP	4A94	"PRINT"
FF5C	HFINP	4AFF	"PRINT"
FF61	HTRMN	4B4D	Erro no "READ" ou "INPUT".
FF66	HFRME	4C6D	Analizador de expressões.
FF6B	HNTPL	4CA6	Analizador de expressões.
FF70	HEVAL	4DD9	Analizador de fatores.
FF75	HOKNO	4F2C	Idem.
FF7A	HFING	4F3E	Idem.
FF7F	HISMI	51C3	"MID\$".
FF84	HWIDT	51CC	"WIDTH".
FF89	HLIST	522E	"LIST".
FF8E	HBUFL	532D	Analizador de tokens.
FF93	HFRQI	543F	Converte para inteiro.
FF98	HSCNE	5514	Apontador do número da linha.
FF9D	HFRET	67EE	Descritor temporário.
FFA2	HPTRG	5EA9	Pesquisador de variáveis.
FFA7	HPHYD	148A	PHYDIO do BIOS.
FFAC	HFORM	148E	FORMAT do BIOS.
FFB1	HERRO	406F	Manipulador de erros.
FFB6	HLPTO	085D	LPTOUT do BIOS.
FFBB	HLPTS	0884	LPTSTT do BIOS.
FFC0	HSGRE	79CC	"SCREEN"
FFC5	HPLAY	73E5	"PLAY"

A área acima de &HFFCA não é usada pelo sistema.

APÊNDICE IV - MONITOR ASSEMBLER

Para que você possa utilizar bem todas as listagens deste livro, fornecemos aqui um programa monitor para auxiliá-lo na entrada e verificação dos dados. O monitor é uma adaptação do programa "Disassembler" publicado no "Aprofundando-se no MSX". Agora, além de disassemblar a memória, pode-se manipulá-la de uma forma mais eficiente e poderosa.

Os novos comandos do monitor são:

D XXXX: lista o conteúdo da memória em hexadecimal a partir do endereço XXXX. O valor XXXX é um número em decimal (para entrá-lo em HEX, use &H). A listagem prossegue pressionando-se qualquer tecla. ESC aborta.

M XXXX: edita os bytes da memória a partir de XXXX. A tecla RETURN avança um byte, BS retorna e ESC aborta. Para alterar o byte corrente, entre um número em HEX (00 a FF).

L XXXX: lista o conteúdo da memória a partir de XXXX em mnemônicos Z80 (disassembler). ESC aborta.

A digitação do monitor pode ser bem simplificada se você já possui o programa Disassembler gravado. Basta renumerá-lo, alterar as linhas iniciais e incluir as linhas restantes do monitor (figura IV.1). Recomenda-se fazer uma boa gravação do monitor pois este será muito útil durante o decorrer do livro. Se você não possui o Disassembler, digite todo o programa da figura IV.1

FIGURA IV.1 - Monitor Assembler

```
1000 CLS:PRINT "Monitor MSX 1986":PRINT
"Milton Maldonado Jr."
1010 PRINT:LINE INPUT ">";A$:IF LEN(A$)
=0 THEN 1010
1020 C$=MID$(A$,1,1):IF C$="L" THEN 113
0 ELSE IF C$="D" THEN 1030 ELSE IF C$="
M" THEN 1060 ELSE BEEP:PRINT "?";:GOTO
1010
1030 E=VAL(MID$(A$,2))
1040 FOR J=1 TO 8:PRINT RIGHT$("0000"+H
EX$(E),4);" ";:FOR I=0 TO 7:PRINT RIGH
T$("00"+HEX$(PEEK(I+E)),2);" ";:NEXT I:
PRINT:E=E+8:NEXT J
```

```

1050 I$=INPUT$(1):IF I$=CHR$(27) THEN 1
010 ELSE PRINT:GOTO 1040
1060 E=VAL(MID$(A$,2))
1070 PRINT:PRINT RIGHT$("0000"+HEX$(E),
4);" ";RIGHT$("00"+HEX$(PEEK(E)),2);"
"
1080 H$="":FOR I=1 TO 2
1090 I$=INPUT$(1):IF I=1 THEN IF I$=CHR
$(13) THEN E=E+1:GOTO 1070 ELSE IF I$=C
HR$(8) THEN E=E-1:GOTO 1070
1100 IF I$=CHR$(27) THEN PRINT:GOTO 101
0
1110 IF I$>="0" AND I$<="9" OR I$>="A"
AND I$<="F" THEN 1120 ELSE 1090
1120 H$=H$+I$:PRINT I$;:NEXT I:POKE E,V
AL("&H"+H$):E=E+1:GOTO 1070
1130 E=VAL(MID$(A$,2))
1140 RESTORE:FOR I=0 TO 7:READ A$(I),B$
(I):NEXT I:FOR I=0 TO 3:READ C$(I),E$(I
):NEXT I:FOR I=0 TO 7:READ D$(I),F$(I):
NEXT I:READ G$(0),G$(1),G$(2),H$(0),H$(
1),H$(2),I$(0),I$(1),J$(0),J$(1)
1150 FOR K=1 TO 20:C=PEEK(E)
1160 PRINT USING "#####";E;:PRINT " ";
:GOSUB 1190:IF F>0 THEN I=I+1
1170 PRINT TAB(22);:FOR X=0 TO I-1:I$=H
EX$(PEEK(X+E)):IF LEN(I$)<2 THEN I$="0"
+I$
1180 PRINT I$;" ";:NEXT X:PRINT:E=E+I:N
EXT K:PRINT:I$=INPUT$(1):IF I$=CHR$(27)
THEN 1010 ELSE 1150
1190 '-----

```

SEPARA GRUPOS

```

-----
1200 F=0:IF C=221 THEN F=1 ELSE IF C=25
3 THEN F=2
1210 C$(2)="HL":IF F=1 THEN C$(2)="IX"
ELSE IF F=2 THEN C$(2)="IY"
1220 IF C=203 THEN 1740 ELSE IF C=237 T
HEN 1800
1230 IF F>0 AND PEEK(E+1)=203 THEN 1740
1240 IF F>0 THEN C=PEEK(E+1)

```

COMANDOS DIRETOS

```

1260 E$(2)=C$(2)
1270 I=1:IF C=39 THEN PRINT "DAA";ELSE
IF C=47 THEN PRINT "CPL";ELSE IF C=249
THEN PRINT "LD SP,";C$(2);ELSE IF C=227
THEN PRINT "EX (SP),";C$(2);ELSE IF C=
118 THEN PRINT "HALT";ELSE IF C=201 THE
N PRINT "RET";ELSE IF C=31 THEN PRINT "
RRA";ELSE 1290
1280 RETURN
1290 IF C=235 THEN PRINT "EX DE,HL";ELS
E IF C=8 THEN PRINT "EX AF,AF";ELSE IF
C=217 THEN PRINT "EXX";ELSE IF C=233 T
HEN PRINT "JP (";C$(2);")";ELSE IF C=0
THEN PRINT "NOP";ELSE IF C=243 THEN PRI
NT "DI";ELSE IF C=251 THEN PRINT "EI";E
LSE 1310
1300 RETURN
1310 IF C=7 THEN PRINT "RLCA";ELSE IF C
=15 THEN PRINT "RRCA";ELSE IF C=23 THEN
PRINT "RLA";ELSE IF C=55 THEN PRINT "S
CF";ELSE IF C=63 THEN PRINT "CCF";ELSE
1330
1320 RETURN
1330 I=2:IF C=211 THEN PRINT "OUT (";:G
OSUB 1610:PRINT ")",A";ELSE IF C=219 THE
N PRINT "IN A,(";:GOSUB 1610:PRINT")";E
LSE 1350
1340 RETURN
1350 IF C=195 THEN PRINT "JP ";ELSE IF
C=205 THEN PRINT "CALL ";ELSE 1380
1360 PRINT MID$(STR$(PEEK (E+1)+256*PEE
K(E+2)),2,5);:I=3:RETURN
1370 RETURN
1380 IF C=16 THEN PRINT "DJNZ ";ELSE IF
C=24 THEN PRINT "JR ";ELSE 1410
1390 X=PEEK(E+1):IF X<128 THEN Y=E+X+2
ELSE Y=E+X-254
1400 PRINT MID$(STR$(Y),2,5);:I=2:RETUR
N

```

```

1410 IF C>127 AND C<192 THEN X=C\8-16:Y
=C MOD 8:PRINT A$(X);:IF X=0 OR X=1 OR
X=3 THEN PRINT " A,";ELSE PRINT " ";ELS
E 1430
1420 PRINT B$(Y);:I=1:RETURN
1430 IF C\64>0 THEN 1570
1440 IF C=34 THEN E=E+SGN(F):PRINT "LD
(",:GOSUB 1360:PRINT ")," :C$(2);:ELSE I
F C=42 THEN E=E+SGN(F):PRINT "LD " :C$(2
);"," :(",:GOSUB 1360:PRINT ")," :ELSE 1460
1450 E=E-SGN(F):RETURN
1460 IF C=50 THEN PRINT "LD (",:GOSUB 1
360:PRINT ")," A";ELSE IF C=58 THEN PRINT
"LD A,(",:GOSUB 1360:PRINT ")," :RETURN
ELSE 1480
1470 I=3:RETURN
1480 X=C MOD 8:IF X=4 THEN PRINT"INC";E
LSE IF X=5 THEN PRINT"DEC";ELSE 1500
1490 Y=C\8:PRINT" " :B$(Y);:I=1:RETURN
1500 X=C MOD 16:IF X=3 THEN PRINT"INC "
;ELSE IF X=11 THEN PRINT"DEC " ;ELSE IF
X=9 THEN PRINT"ADD " :C$(2);"," :ELSE 152
0
1510 X=C\16:PRINTC$(X);:I=1:RETURN
1520 IF C MOD 8=6 AND F=0 THEN PRINT "L
D " :B$(C\8);"," :GOTO 1610
1530 IF C MOD 16=1 THEN E=E+SGN(F):PRIN
T"LD " :C$(C\16);"," :GOSUB 1360 ELSE 15
50
1540 E=E-SGN(F):RETURN
1550 IF C MOD 16=2 THEN PRINT "LD (" :C$
(C\16);")," A";ELSE IF C MOD 16=10 THEN P
RINT "LD A,(" :C$(C\16);")," :ELSE 1570
1560 I=1:RETURN
1570 IF C\32=1 AND C MOD 8=0 THEN PRINT
"JR " :D$(C\8-4);"," :GOTO 1390
1580 IF C\64<>3 THEN 1680
1590 IF C MOD 8<>6 THEN 1620
1600 I=2:X=C\8-24:PRINTA$(X);" " :IF X<
2 OR X=3 THEN PRINT"A,";
1610 PRINT MID$(STR$(PEEK(E+1)),2,3);:R
ETURN

```

```

1620 X=C MOD 8:IF X=1 THEN PRINT "POP "
;E$(C\16-12);ELSE IF X=5 THEN PRINT "PU
SH ";E$(C\16-12);
1630 IF X=1 OR X=5 THEN I=1:RETURN
1640 IF X=7 THEN PRINT"RST ";HEX$(C-199
);"H";:I=1:RETURN
1650 IF X=0 THEN PRINT "RET ";D$(C\8-24
);:I=1:RETURN
1660 IF X=2 THEN PRINT "JP ";ELSE PRINT
"CALL ";
1670 PRINT D$(C\8-24);",";:GOTO 1360
1680 IF F=0 AND C\64=1 THEN X=(C-64)\8:
PRINT "LD ";B$(X);:X=(C-64) MOD 8:PRINT
",";B$(X);:I=1:RETURN
1690 I=2:C=PEEK(E+1):IF F>0 AND C MOD 8
=6 AND C>63 THEN PRINT "LD ";B$(C\8-8);
",";C$(2);"+";:E=E+1:GOSUB 1610:PRINT
")"; ELSE 1710
1700 E=E-1:RETURN
1710 IF C\8=14 THEN E=E+1:PRINT "LD (";
C$(2);"+";:GOSUB 1610:PRINT ");";B$(C-1
12); ELSE 1730
1720 GOTO 1700
1730 I=3:IF C=54 THEN PRINT "LD (";C$(2
);"+";:E=E+1:GOSUB 1610:PRINT ");";:E=E
+1:GOSUB 1610:E=E-1:GOTO 1720 ELSE 1960
1740 '-----

```

COMANDOS APOS CBH

```

-----
1750 Z=1:IF F>0 THEN Z=3
1760 I=2:C=PEEK(E+Z):IF C<64 AND F=0 TH
ENPRINT F$(C\8);" ";B$(C MOD 8);:RETURN
1770 I=3:IF C<64 AND F>0 AND (C-6) MOD
8=0 THEN PRINT F$(C\8);" (";C$(2);"+";:
E=E+1:GOSUB 1610:PRINT")";:E=E-1:RETURN
1780 C=PEEK(E+Z):IF F=0 THEN PRINTG$(C\
64-1);" ";CHR$(48+(C\8) MOD 8);",";B$(C
MOD 8);:I=2:RETURN
1790 IF (C-6) MOD 8=0 THEN PRINTG$(C\64
-1);" ";CHR$(48+(C\8) MOD 8);",";C$(2)
;"+";:E=E+1:GOSUB 1610:PRINT")";:E=E-1:
RETURN ELSE 1960

```

1800 '-----

COMANDOS APOS EDH

1810 I=2:C=PEEK(E+1):IF C<64 OR C=221 O
R C=253 THEN 1960 ELSE IF C>187 THEN 19
20

1820 IF C=70 THEN PRINT "IM 0";ELSE IF
C=86 THEN PRINT "IM 1";ELSE IF C=94 THE
N PRINT "IM 2";ELSE IF C=77 THEN PRINT
"RETI";ELSE IF C=69 THEN PRINT "RETN";E
LSE IF C=103 THEN PRINT "RRD";ELSE IF C
=111 THEN PRINT "RLD";ELSE 1840

1830 RETURN

1840 IF C=71 THEN PRINT "LD I,A";ELSE I
F C=79 THEN PRINT "LD R,A";ELSE IF C=87
THEN PRINT "LD A,I";ELSE IF C=95 THEN
PRINT "LD A,R";ELSE 1860

1850 RETURN

1860 H\$(3)="OUT":IF C>175 THEN H\$(3)="O
T"

1870 IF C>159 AND C MOD 8<4 THEN PRIN
TH\$(C MOD 4);I\$((C MOD 16)\8);J\$((C-160)\
16);:RETURN

1880 IF C\64>1 THEN 1960 ELSE IF C MOD
8>1 THEN 1920

1890 IF C=112 OR C=113 THEN 1960

1900 IF C MOD 8=0 THEN PRINT "IN ";B\$(C
\8-8);", (C)";ELSE PRINT "OUT (C),";B\$(C
\8-8);

1910 RETURN

1920 X=C MOD 16:Y=C\16-4:IF X=10 THEN P
RINT"ADC HL,";C\$(Y);ELSE IF X=2 THEN PR
INT "SBC HL,";C\$(Y);ELSE 1940

1930 RETURN

1940 IF X=3 THEN PRINT "LD (";:E=E+1:GO
SUB 1360:E=E-1:PRINT ")," ;C\$(C\16-4);EL
SE IF X=11 THEN PRINT "LD ";C\$(C\16-4);
", (" ;:E=E+1:GOSUB 1360:E=E-1:PRINT")";E
LSE 1960

1950 I=4:RETURN

1960 PRINT "Z80 ?";:RETURN

1970

DADOS DA MATRIZ ALFA

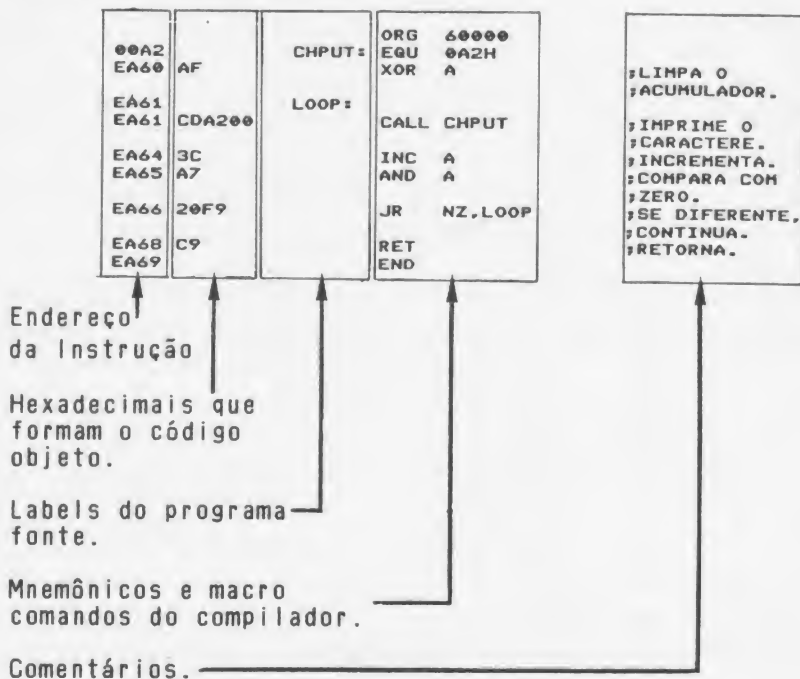
1980 DATA ADD,B,ADC,C,SUB,D,SBC,E,AND,H
 ,XOR,L,OR,(HL),CP,A
 1990 DATA BC,BC,DE,DE,HL,HL,SP,AF
 2000 DATA NZ,RLC,Z,RRC,NC,RL,C,RR,PO,SL
 A,PE,SRA,P,SLI,M,SRL
 2010 DATA BIT,RES,SET
 2020 DATA LD,CP,IN,I,D, ,R

DIGITANDO UMA LISTAGEM EM ASSEMBLY

Vamos ver agora um exemplo de como entrar uma listagem do livro na memória para executá-la. Para isto, carregue o monitor na memória e digite RUN. Deverá aparecer uma apresentação e o sinal ">", indicando que o monitor aguarda um comando.

Observe na figura IV.2 o formato das listagens dos programas deste livro.

FIGURA IV.2 - Exemplo de Listagem



Para entrá-lo no monitor, digite:

M 60000

ou

M &HEA60

O monitor deverá responder:

EA60 00

Digite, então, o primeiro hexadecimal que forma o código objeto (AF), e então teremos:

EA60 00 AF

EA61 00

Continue, então, com CD, A2, 00, etc e a tela deverá ficar como mostra a figura IV.3

FIGURA IV.3 - Exemplo de uso do monitor

EA60	00	AF
EA61	00	CD
EA62	00	A2
EA63	00	00
EA64	00	3C
EA65	00	A7
EA66	00	20
EA67	00	F9
EA68	00	

Ao final do processo, digite ESC para sair do modo M. Para testar o programa, entre CTRL+STOP e execute-o comandando (no caso específico deste programa)

DEFUSR=60000#X=USR(0)..

Se você possuir um Assemblador como o Coral ASM ou o HOTASM, deverão ser digitados os números das linhas (característica que as listagens do livro não possuem), os labels, os mnemônicos ou macrocomandos e se você desejar os comentários.

Caso o seu compilador possua sintaxe semelhante à do M80, devem ser digitados apenas os mnemônicos ou macrocomandos e, se você desejar, os comentários.

OUTRAS PUBLICAÇÕES SOBRE MSX DA EDITORA ALEPH

APROFUNDANDO-SE NO MSX

Piazzzi, Maldonado, Oliveira et al.- Para quem quer conhecer todos os detalhes da máquina: como usar os 32Kb de RAM escondidos pela ROM, como redefinir caracteres, como usar o SOUND, como tirar cópias de telas gráficas na impressora, como destravar fitas. Todos os detalhes da arquitetura do MSX, o BIOS e as variáveis do sistema comentadas e um poderoso programa Disassembler.

TABELA DE MNEMÔNICOS Z80

Uma tabela com todos os Mnemônicos do microprocessador Z80 (o microprocessador do MSX) relacionados com seus códigos hexadecimais. Indispensável para quem está começando a programar em linguagem de máquina e não dispõe de um programa Assembler.

USANDO O DISK DRIVE NO MSX

Rubens Pereira Silva Jr- O DISK BASIC, o MSX D.O.S. e o CP/M para o MSX comentados detalhadamente. Desde a instalação até os usos mais sofisticados. Completando a obra, um dicionário de todos os comandos e funções desses três sistemas operacionais. Indispensável para quem tem um drive !

SISTEMA DE DISCO PARA MSX

Oliveira e Pereira- Como instalar a interface CDX-2 utilizando o sistema operacional residente em sua ROM, o poderoso BASIC DE DISCO. Como tirar o máximo proveito do SOLX-DOS, sistema operacional totalmente compatível com o MSX D.O.S. e semi-compatível com o CP/M. Como converter programas instalados em CP-500 e em sistema 700 para o MSX e vice-versa. Complementando a obra, um detalhado dicionário de todos os comandos e funções, repleto de exemplos didáticos e aplicações. Uma obra importantíssima para quem quer tirar o máximo proveito de seu sistema MSX!

COLEÇÃO DE PROGRAMAS PARA MSX VOL. I

Oliveira et al.- Uma coletânea de programas para o usuário principiante em MSX. Jogos, músicas, desenhos e aplicativos úteis apresentados de modo simples e didático. Todos os programas têm instruções de digitação e uma análise detalhada, explicando praticamente linha por linha o seu funcionamento. Todos os programas foram testados e funcionam! A maneira mais fácil e divertida de entrar no maravilhoso mundo dos micros MSX.

COLEÇÃO DE PROGRAMAS PARA MSX VOL. II
Oliveira et al.- Programas com rotinas em BASIC e Linguagem de Máquina. Jogos de ação e inteligência, programas didáticos, programas profissionais de estatística, matemática financeira e desenho de perspectivas, utilitários para uso da impressora e gravador cassete. E ainda um capítulo especial mostrando como montar, passo a passo, um jogo de ação, o ISCAI JEGUE, uma paródia bem humorada do famoso SKY JAGUAR!

JOGOS DE HABILIDADE PARA MSX

Dias, Guazelli, Martins e Sung- Coletânea de jogos de ação e inteligência de nível profissional, usando sempre que necessário, os recursos da Linguagem de Máquina. Asteróides, Coelho Maluco, Alcatraz, Gincana, são alguns jogos listados neste livro com suas instruções de como utilizá-los. A maneira mais econômica de obter software profissional de alta qualidade, divertindo-se ao montar uma vasta biblioteca de programas.

LINGUAGEM BASIC MSX

Uma verdadeira enciclopédia do BASIC MSX onde todos os comandos e funções do BASIC residente são listados em ordem alfabética. Para cada comando é dada a explicação do que ele faz, sua sintaxe e um exemplo elucidativo. Obra de consulta, indispensável para quem programa em BASIC MSX, adotada pela GRADIENTE para acompanhar seu microcomputador Expert.

HOTLOGO - Primeiros Passos

Godoy, Lacerda, Lepíscopo e Mendes- Livro dedicado às crianças que começam a entrar no mundo maravilhoso da computação. A Linguagem utilizada é o HOTLOGO, disponível em cartucho para MSX, uma das versões mais completas da linguagem LOGO.

HOTDATA - Gerenciador de Dados

Roberto Massaru Watanabe- O gerenciamento de um banco de dados explicado de uma maneira didática para quem utiliza o cartucho HOTDATA para MSX.

HOTWORD - Processador de Textos

Roberto Massaru Watanabe- O processamento de textos no micro, explicado de uma maneira didática para quem usa o cartucho HOTWORD para MSX.

HOTPLAN - Planilha de Cálculos

Roberto Massaru Watanabe- Uma das melhores planilhas eletrônicas, explicada com exemplos e figuras para os que utilizam o cartucho HOTPLAN para HOTBIT.

DOMINANDO O EXPERT

Os primeiros passos na programação e utilização do microcomputador Expert com explicações didática para os usuários principiantes. Obra adotada pela GRADIENTE.

RESUMO DE OPERAÇÕES DO EXPERT

Tabela de consulta rápida com todos os comandos e funções do BASIC MSX. Complemento extremamente útil da obra "LINGUAGEM BASIC MSX" para ser deixado constantemente ao lado do teclado de seu MSX.

Para receber mais informações sobre nossos livros e ganhar uma assinatura gratuita de nosso boletim informativo, contendo dicas e programas para MSX, envie seu nome e endereço completos (inclusive CEP!) para

Editora Aleph
Caixa Postal 20.707
01498 São Paulo SP

A composição deste livro foi feita numa Mônica Plus 6030 da ELEBRA, utilizando-se fontes alternativas para MSX. Esta mesma impressora foi utilizada para impressão de todas as listagens e figuras utilizando os programas de cópia gráfica descritos neste livro e no "A-PROFUNDANDO-SE NO MSX" desta editora.

elebra  **informática**

Gráfica Palas Athena
Associação "Palas Athena" do Brasil
Rua José Bento, 384
Fone: 279-6288 — CEP 01523
Cambuci — São Paulo

COLEÇÃO MSX



PROGRAMAÇÃO AVANÇADA EM MSX

Este é um livro para aqueles que gostam de extrair de uma máquina tudo o que ela tem a oferecer.

Todos os segredos do hardware dos micros MSX são comentados e os truques e macetes para controlá-los através da Linguagem de Máquina do Z80 são exaustivamente ensinados.

Os autores, Henrique, Milton e Paulo, que já trabalharam juntos no "APROFUNDANDO-SE NO MSX", chegam agora ao limite de conhecimento desses micros, mantendo contudo, a mesma clareza e inteligibilidade de seus textos anteriores.

Esta é mais uma obra indispensável na biblioteca e (principalmente) na mente dos programadores que querem usar todos os recursos dos micros MSX!

HENRIQUE DE FIGUEREDO LUZ

MILTON MALDONADO JR.

PAULO EDUARDO GUIMARÃES ROSSETTO

